FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA

SOLVING TRANSCENDENTAL EQUATION USING GENETIC ALGORITHM

MASITAH HAMBARI
WEK 020119
2004/2005

SUPERVISOR
ASS. PROF. DR. SAMEEM ABD KAREEM

MODERATOR
ASS. PROF. DR. ROZIATI ZAINUDDIN

ABSTRACT

A problem of great importance in engineering is finding the roots of the equations. There are closed form expressions for the case of linear, quadratic or cubic equations. For higher orders like Transcendental Equation, Numerical Methods is used to find the solutions. Genetic algorithm (GA) has long been used for optimization problems that arise in a wide variety of complex systems. This project studies and explores the potential of using Genetic Algorithm to solve Transcendental Equation. Genetic Algorithm is used to find the roots or set of optimal solution that satisfy the equation. The most critical task for developing this project is how to encode the algorithm based on the domain, Transcendental Equation. This project studies the potential of using Genetic Algorithm and develops a program in order to solve Transcendental Equation in optimizing problem domain. This solution hopefully may provide a potential alternative for a better solution.

ACKNOWLEDGEMENT

First and foremost, I would like to express my greatest gratitude to my project supervisor, Associate Professor Dr Sameem Abdul Kareem was has been patient and kind through the development of this project. Her continues guidance has helped me a lot in determining how the project should take shape, and her willingness to listen and help me all the way is much appreciated. In addition, I would like to thank my project moderator, Associate Professor Dr Roziati Zainuddin, for her constructive comments during my project proposal presentation. Last but not least, I would like to thank to all my friends and course mates for offering ideas, opinions and support me during the implementation of the project. The completion of this project will not be accomplished without the assistance from all these people.

TABLE OF CONTENT

LIST OF FIGURE

LIST OF FIGURE

LIST OF TABLE

CHAPTER I

INTRODUCTION

CHAPTER I INTRODUCTION

1.1 INTRODUCTION

Genetic algorithms are computational procedures that mimic the natural process of evolution. The theories of evolution and natural selection, proposed by Darwin to explain the concept of natural selection was used to explain how species have been able to adapt to changing environments and evolved.

The objective of the Genetic Algorithm is then to find an optimal solution to a problem. Since Genetic Algorithm are heuristic procedures, they are not guaranteed to find the optimum, but at least they are able to find the best solutions for a wide range of problems.

In this project we intend to develop an application to solve Transcendental Equations using Genetic Algorithms (GAs). Transcendental equation is an equation or formula involving transcendental function, which is not an algebraic function. It contains non-algebraic expressions – exponential, trigonometric, logarithmic and other functions. Based on GAs success in problem solving we believe that we would be able to develop a system that employs GA to find the root of a Transcendental Equation that is efficient and fast.

1.2 PROJECT MOTIVATION

Traditionally, many typical methods are used for solving Transcendental Equations (Chapter II). However these methods work inefficiently in many cases. Infact, none of this method seeks to find all solutions of a given transcendental equation in a given range. This is a big drawback because many physical problems yielding transcendental equations have more than a single solution. In such cases, exhaustive search remains the only option when using traditional root finding methods.

The advantage of the GA approach is the ease with which it can handle many kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA scheduler to the particular requirements of a very wide range of possible overall objectives.

1.3 OBJECTIVE

The aim of this project is to build a system, which would solve a dedicated Transcendental Equation using the Genetic Algorithm approach. Genetic Algorithm is applied to find the roots of the equation, a set of optimal solution that satisfies the equation. Furthermore, the system will be able to monitor the performance of Genetic Algorithm applied, and provides a modeling of the solution.

1.4 SCOPE

The scope of the project is to develop a Genetic Algorithm application in order to solve Y = 5 cos (x) . The application will be developed using MATLAB.

1.5 PROJECT DEVELOPMENT METHODOLOGY

This project will be developed using Waterfall Life-Cycle Model, in which the Development begins from scratch, then followed by requirement elicitation, analysis, design and finally the implementation. The Waterfall Life-Cycle Model provides a step-by-step guidance, which makes project development systematic and easy to follow and understand. This system development life-cycle model is suitable for the project as it is dedicated for small-scale projects.

1.6 SUMMARY

The general ideas and objectives of this project are discussed in this introduction chapter. Chapter 1 (Introduction) gives an overview of the project together with its, objectives, scope and function. Chapter II (Literature Review) explains the background studies carried out in order to build the system. It includes details of the field of the studies involved. A description of the methodology, planning and techniques used in the project is discussed in Chapter III (Methodology).

The project requirements such as functional requirement, non-functional requirement, hardware and software needed with respect to the project is discussed in Chapter IV (System Analysis). Chapter V (System Design) describes the design phase of the project, which includes the design of the modules, development tools used and the project requirements.

CHAPTER II

LITERATURE REVIEW

CHAPTER II LITERATURE REVIEW

2.1 INTRODUCTION

In this chapter, we discuss the background studies that lead us to use genetic algorithm as a method to solve transcendental algorithm and eventually to build our prototype. We start with a description of genetic algorithm, followed by transcendental equation, then analysis of some methods of solving transcendental equation, disadvantage of some methods of solving transcendental equation, genetic algorithm as alternative methods to solve transcendental equation, introduction to MATLAB as a tool for optimization problem and end up with summary. We also include some of the recent work done by other researchers in the area of using GA to solve transcendental equations. Details information of the field of the studies involved are also included.

2.2 GENETIC ALGORITHM

2.2.1 Introduction

Genetic Algorithms (GAs) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and population genetic. The basic concept of GAs is designed to simulate processes in natural system necessary for evolution [20]. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

First pioneered by John Holland in the 60s, Genetic Algorithms has been widely studied, experimented and applied in many fields in engineering worlds [1]. Not only does GAs provide an alternative method to solving problem, it consistently outperforms other traditional methods in most of the problems link [2]. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for GAs

GAs uses a set of possible solutions called population to solve problems. They do not require that a whole data set be used in the operation [2]. In addition, they are able to produce robust systems that adapt well to changes within the problem parameters. GAs is a form of randomized search in the sense that in GAs strings are chosen and combined in a stochastic process. This is how the GA approach is

radically different to the problem solving methods used by more traditional algorithms, which are deterministic in nature.

2.2.2 Who Can Benefit From GA

Nearly everyone can gain benefits from Genetic Algorithms, once he can encode solutions of a given problem to chromosomes in a GA, and compare the relative performance (fitness) of the solutions. An effective GA representation and meaningful fitness evaluation are the keys of the success in GA applications [17]. GAs are useful and efficient when

- The search space is large, complex or poorly understood.
- Domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space.
- No mathematical analysis is available.
- Traditional search methods fail.

2.2.3 How GA work

Genetic Algorithm codes parameters of the search space as binary strings of fixed length. It employs a population of strings initialized at random, which evolve to the next generation by genetic operators such as selection, crossover and mutation [23]. The fitness function evaluates the quality of solutions coded by strings.

Selection allows strings with higher fitness to appear with higher probability in the next generation. Crossover combines two parents by exchanging parts of their strings, starting from a randomly chosen crossover point. This leads to new solutions inheriting desirable qualities from both parents.

Mutation flips single bits in a string, which prevents the GA from premature convergence, by exploiting new regions in the search space. GA tends to take advantage of the fittest solutions by giving them greater weight, and concentrating the search in the regions, which lead to fitter structures, and hence better solutions of the problem.

Finding good parameter settings that work for a particular problem is not a trivial task. The critical factors are to determine robust parameter settings for population size, encoding, selection criteria,

genetic operator probabilities and evaluation (fitness) normalization techniques [2].

- If the population is too small, the genetic algorithm will converge too quickly to a local optimal point and may not find the best solution. On the other hand, too many members in a population result in a long waiting time for significant improvement.
- Coding the solutions is based on the principle of meaningful building blocks and the principle of minimal alphabets, by using the binary strings.
- The fitter member will have a greater chance of reproducing. The members with lower fitness are replaced by the offspring. Thus in successive generations, the members on average are fitter as solutions to the problem.
- Too high mutation introduces too much diversity and takes longer time to get the optimal solution. Too low mutation tends to miss some near-optimal points. Two-point crossover is quicker to get the same results and retain the solutions much longer than one point crossover.

The fitness function links the Genetic Algorithm to the problem to be solved. The assigned fitness is used to calculate the selection probabilities for choosing parents, for determining which member will be replaced by which child.

2.2.4 GA Learning Process

GAs exploits the idea of survival of the fittest and interbreeding population to create a novel and innovative search strategy [1]. GA maintains a population of strings, representing solution to a specified problem. Then GA iteratively creates new population from other old by ranking the string and interbreeding the fittest to create new strings, which is hopefully closer to the optimum solution to the problem at hand.

A population of solution is randomly generated and each solution is evaluated for its fitness, which is a measure directly related to the solution performance in the problem task. Thus, high-fitness or 'good-individuals' stand a better chance of 'reproducing' than the low-fitness are more likely to disappear.

In such algorithms of a population of individual (potential solution) undergoes a sequence of operators like mutation and crossover transformation. These individual strive for survival, a selection (reproduction) scheme biased towards selecting fitter individuals, produces the individual for the next generation. After some number of generations, the program converge the best individual represents the optimum solution.

The idea of survival of the fittest is the important feature to GA [1]. It used fitness function in order to select the fittest string that will be use to create new, conceivably better populations strings. The only thing that the fitness function must do is to rank the string in some way by producing the fitness value, which is then are used to select fittest string. The concept of fittest string is, in fact, a particular instance of a more general concept of Artificial Intelligence, the objective function.

2.2.5 Feature of GA

GA consists of several features that construct the whole architecture of GA algorithm [5].

- Parameter Set: *Random choices of parameter, which is to be the fit function.*
- Fitness of Parameter Set: *A calculation of how well a parameter set fits.*
- Citizen: *A collection of parameter set, its binary string and its fitness.*
- Population: *A collection of citizen.*
- Parent: *Initial state of citizen.*
- Children: *A new complete set of of binary string that is taken from two good parents through a mating process.*

- Selection Process: *A process of selecting parents with a good fit (good parent) to breed.*
- Genetic Operators: *A certain operator that is applied to a population to evolve the solution in order to find the best one.*

2.2.6 GA Basic Operation

2.2.6.1 Reproduction/Selection

The reproduction operator allows individual strings to be copied for possible inclusion in the next generation. The chance that a string will be copied is based on the string's fittest value, calculated from a fitness function. For each generation, the reproduction operators choose strings that are placed in the mating pool, which is used as the basis for the next generation. There are many different type of reproduction operators. One always selects the fittest and discards the worst. There are hundred of variants of this scheme. None are right or wrong. In fact, some will perform better than others depend on the problem domain.

2.2.6.2 Crossover

Remember that crossover is biological term that refers to the blending of chromosome from the parents to produces new chromosome. The GAs select two strings at random from the matting pool. The strings may be different or identical, but it does not matter. Then, it will calculate whether the crossover operators should take place using a parameter called crossover probability. If GAs decides not to perform the crossover operator, the two-selected strings are simply copied to the new population. If crossover does take place, then a random splicing point is chosen in the string, the two strings are spliced and the spliced regions are mixed to create two potentially new strings. The child strings are then placed in the new population.

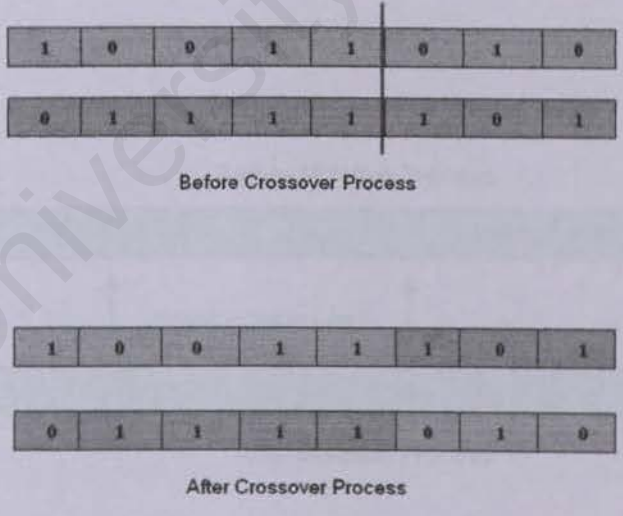


Figure 2.1: Crossover Process

2.2.6.3 Mutation

Selection and crossover alone can generate staggering amounts of variant strings. However, depending on the initial population chosen, there may not be enough variety of strings to ensure the GAs sees the entire problem space. Or the GAs may find itself converging on strings that are not quite close to the optimum it seek due to a bad initial population. Mutation operators cover some of these problems. The GAs has a mutation probability, which dictates the frequency in which mutation occurs. Mutation can be perform either during selection or crossover, but usually perform during crossover. For each string's element, in each string in the meeting pool, the GAs checks to see if it should perform mutation. If mutation is performed, the element value is randomly changed to a new one.

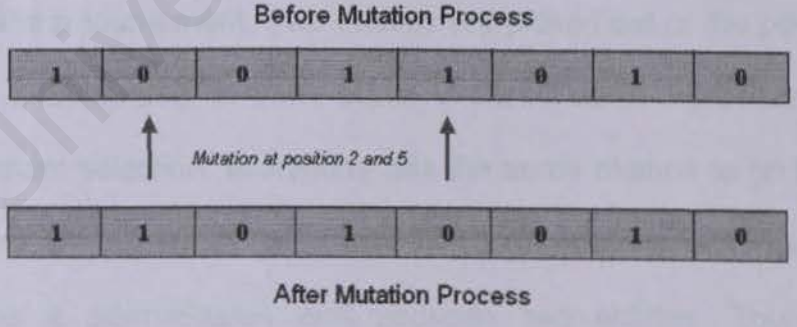


Figure 2.2: Mutation Process

2.2.7 GA Selection Process

2.2.7.1 Truncation Selection

Truncation selection is a selection method used in genetic algorithms to select potential candidate solutions for recombination [5]. In truncation selection the candidate solutions are ordered by fitness, and some proportion, p, (e.g. p=1/2, 1/3, etc.), of the fittest individuals are selected and reproduced 1/p times. However, using this method, it will eliminate a fixed percentage of the weakest candidates. Truncation selection is less sophisticated than many other selection methods, and is not often used in practice.

2.2.7.2 Rank Tournament Selection

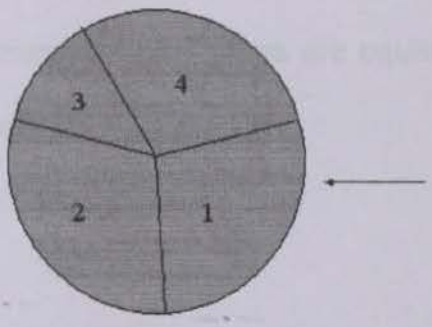
In tournament selection, the entities that are allowed to reproduce are chosen in a tournament. Two entities are picked out of the pool, their fitness is compared, and the better is permitted to reproduce [5]. In tournament selection, everybody has the same chance to go into the tournament. Therefore, the fitness function does not really matter as long as it discriminates well between two entities. Tournament selection is able to overcome the problem of finding a good fitness function for every single problem.

2.7.7.3 Roulette Wheel Selection

The analogy of Roulette-Wheel selection to a roulette wheel can be envisaged by imagining a roulette wheel in which each candidate solution represents a pocket on the wheel; the size of the pockets are proportionate to the probability of selection of the solution [5]. Selecting N chromosomes from the population is equivalent to playing N games on the roulette wheel, as each candidate is drawn independently.

In Roulette-Wheel selection, as in all selection methods, possible solutions or chromosomes are assigned fitness by the fitness function. In this method, this fitness level is used to associate a probability of selection with each individual chromosome.

While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be. With Roulette-Wheel selection, there is a chance some weaker solutions may survive the selection process. This is an advantage, as though a solution may be weak, it may include some component, which could prove useful following the recombination process [22].



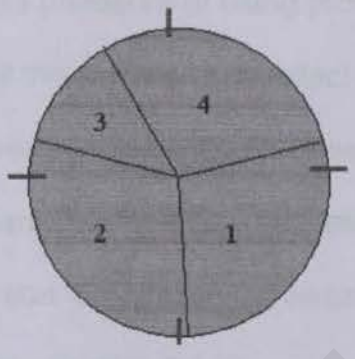
Spin 1 Chromosome 2 selected
Spin 2 Chromosome 1 selected
Spin 3 Chromosome 2 selected
Spin 4 Chromosome 4 selected

Figure 2.3: Roulette Wheel Selection Model

2.7.7.4 Stochastic Universal Selection

Another name for this scheme is Stochastic Sampling without Replacement, was developed by De Jong to minimize the stochastic errors that can result from Roulette Wheel Selection [22]. Using this selection, individuals are mapped to a contiguous line segment similar to that of the Roulette Wheel Selection.

In Stochastic Universal search, equally spaced pointers are position over line/wheel. The numbers of pointers are equal to the number of individual to be selected.



Chromosome 1 1 copy
Chromosome 2 2 copies
Chromosome 3 0 copies
Chromosome 4 1 copy

Figure 2.4: Stochastic Universal Selection Model

2.2.8 How To Use GA

The basic steps involved in a GA are the following:

1. Build an initial population of samples (solutions) created randomly or using some initialization method.
2. Calculate the fitness (measure of being provided reproductive opportunities) of all the samples and select individuals for the reproduction process. The selection is based on fitness, but it is a probabilistic mechanism. Roulette wheel selection, Rank Tournament Selection, Stochastic Universal Selection are some of the selections used. (Section 2.1.7)
3. Apply the genetic operators of crossover, mutations, inversions, etc. to the selected individuals to create new individuals and thus a new generation. In GAs crossovers 'explore' around the already found good solutions and mutations help 'exploiting' the search space for new solutions.
4. Then again Step2 is followed till the condition for ending the algorithm is reached.

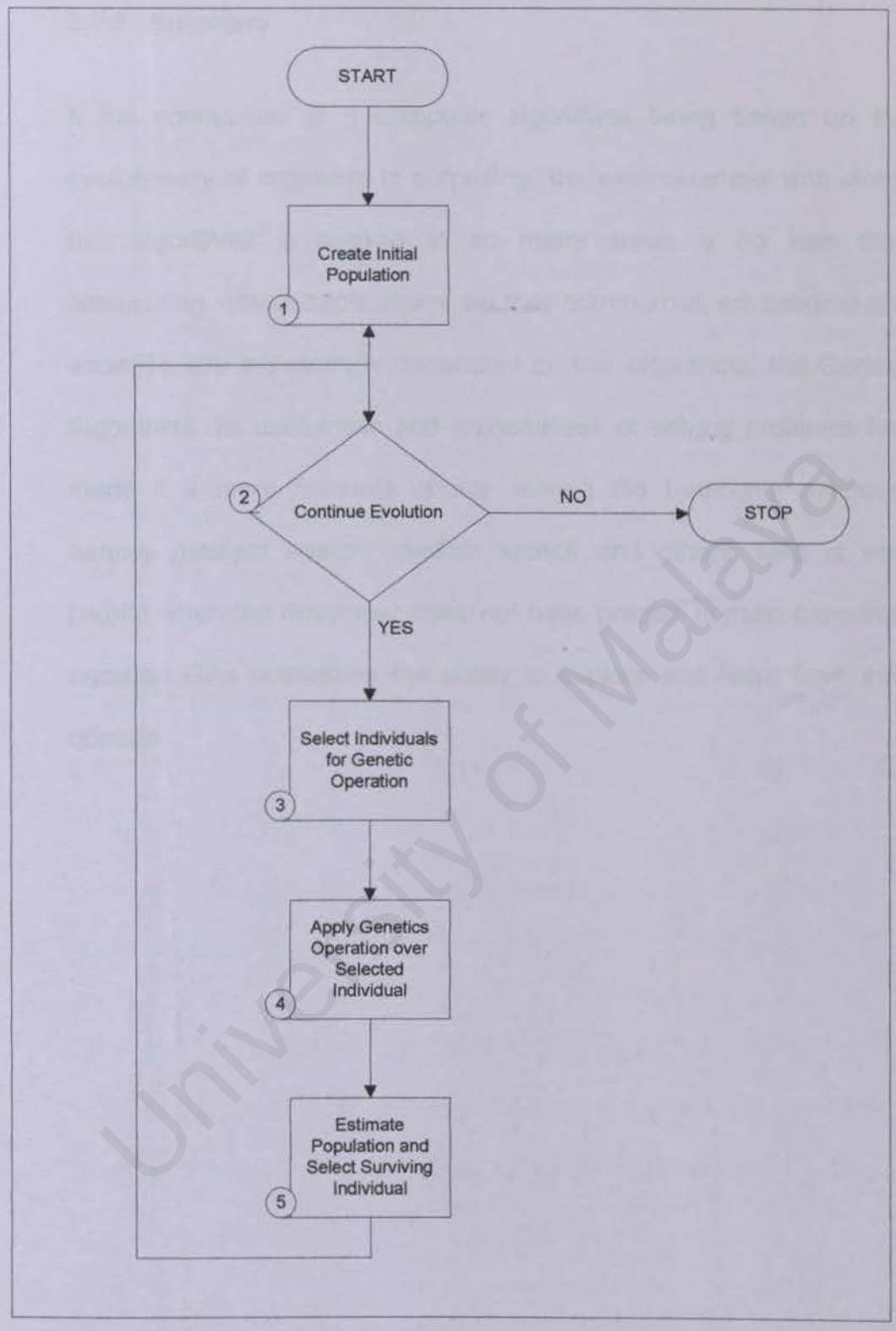


Figure 2.5: Genetic Algorithm Flowchart

2.2.9 Summary

If the conception of a computer algorithms being based on the evolutionary of organism is surprising, the extensiveness with which this algorithms is applied in so many areas is no less than astonishing. These applications, be they commercial, educational and scientific, are increasingly dependent on this algorithms, the Genetic Algorithms. Its usefulness and gracefulness of solving problems has made it a more favourite choice among the traditional methods, namely gradient search, random search and others. GAs is very helpful when the developer does not have precise domain expertise, because GAs possesses the ability to explore and learn from their domain.

2.3 TRANSCENDENTAL EQUATION

2.3.1 Introduction

Transcendental equation is an equation or formula involving transcendental function [9]. Transcendental function is a function, which is not an algebraic function. In other words, a function, which "transcends" cannot be expressed in terms of algebra [16]. It contains non-algebraic expressions – exponential, trigonometric, logarithmic and other functions.

These functions are called transcendental because they cannot be defined directly by algebraic formulas. This means that the only way to work with these functions is to learn to use their algebraic and geometric properties. A Transcendental Curve illustrates a transcendental function.

Example of Transcendental Equation is Exponential Function. The exponential function is the entire function defined by exp(z) = e power(z) where *e* is the constant 2.718. It satisfies the identity exp x+Y) = exp(x) exp(y).

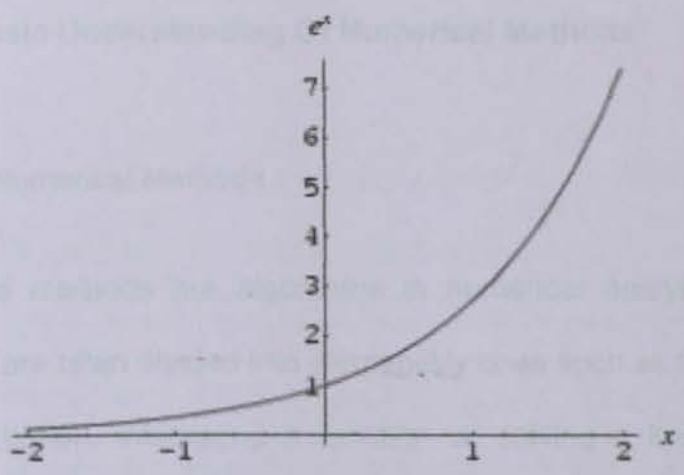


Figure 2.6: Two Dimensional Exponential Graphs

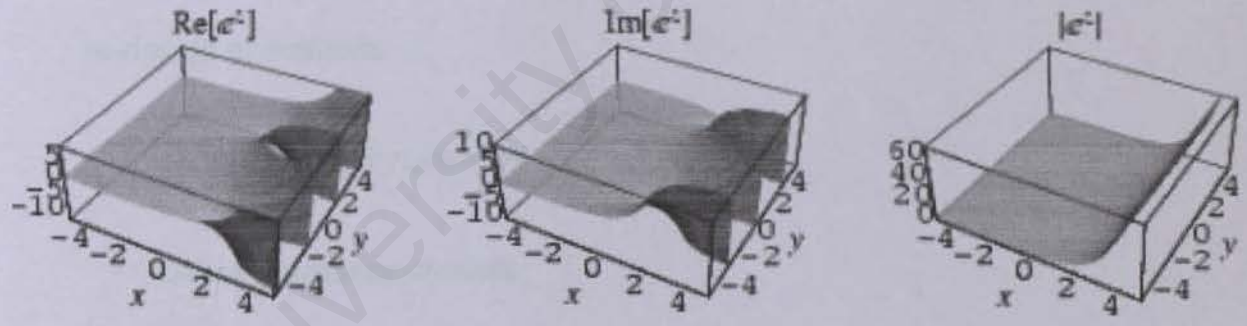


Figure 2.7: Three Dimensional Exponential Graphs.

2.3.2 Basic Understanding Of Numerical Methods

2.3.2.1 Numerical Methods

Numerical methods are algorithms in numerical analysis. Numerical methods are often divided into elementary ones such as finding the root of an equation, integrating a function or solving a linear system of equations to intensive ones [15]. Numerical methods do not usually give the exact answer to a given problem, but they can only tend towards a solution getting closer and closer with each iteration.

The study of the behavior of numerical methods is called numerical analysis. This is a mathematical subject that considers the modeling of the error in the processing of numerical methods and the subsequent re-design of methods.

Some applications of numerical methods are [15]:

- *Linear regression models*
- *Curve fitting*
- *Scatter plots*
- *Approximating functions*
- *Function tables*
- *Scientific computing*

2.3.2.2 Roots of Equations

The value of x which makes f(x)= 0 are called roots or 'zeros' of the equation. For quadratic equation roots can be found by a standard formula. But roots finding for Transcendental Equation are exhausting search. Two types of problems would be deal are real roots of algebraic and transcendental equations and complex roots of polynomials.

2.3.2.3 Methods for finding the roots

1. Graphical Methods
2. Bracketing Methods
   - Bisection Method
   - False Position Method – (Regula Falsi)
3. Open Methods
   - Fixed point iteration
   - Newton-Raphson Method
   - Secant Method
4. Multiple roots
5. Systems of Non-linear Equations

2.4 SOME COMMON METHOD OF SOLVING TRANSCENDENTAL EQUATION

2.4.1 Bisection Method

This is the simplest method for finding a root to an equation [9]. One of the main drawbacks is that we need two initial guesses x_a and x_b which bracket the root: let $f_a = f(x_a)$ and $f_b = f(x_b)$ such that $f_a f_b <= 0$. Clearly, if $f_a f_b = 0$ then one or both of x_a and x_b must be a root of $f(x) = 0$.

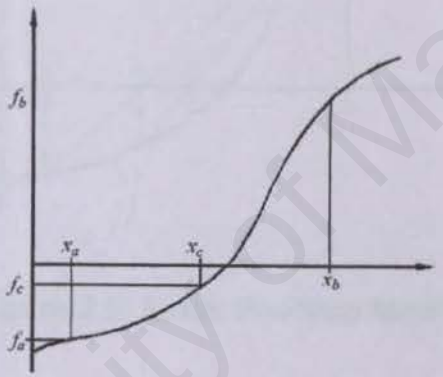


Figure 2.8: Bisection Model

The basic algorithm for the bisection method relies on repeated application of

- Let $x_c = (x_a+x_b)/2$,
- if $f_c = f(c) = 0$ then $x = x_c$ is an exact solution,
- elseif $f_a f_c < 0$ then the root lies in the interval (x_a,x_c),
- else the root lies in the interval (x_c,x_b).

2.4.2 False Position Method (Linear Interpolation)

This method is similar to the bisection method in that it requires two initial guesses to bracket the root [9]. However, instead of simply dividing the region in two, a linear interpolation is used to obtain a new point, which is closer to the root than the equivalent estimate for the bisection method.

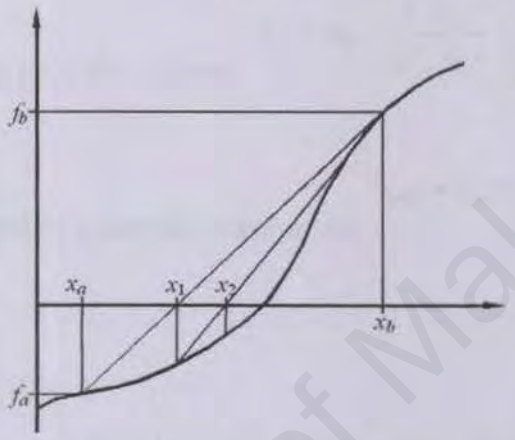


Figure 2.9: False Position Model

The basic algorithm for the linear interpolation method is

- Let $x_c = x_a - \frac{x_b - x_a}{f_b - f_a} f_a = x_b - \frac{x_b - x_a}{f_b - f_a} f_b = \frac{x_a f_b - x_b f_a}{f_b - f_a}$, then
- if $f_c = f(x_c) = 0$ then $x = x_c$ is an exact solution,
- elseif $f_a f_c < 0$ then the root lies in the interval (x_a, x_c),
- else the root lies in the interval (x_c, x_b).

Because the solution remains bracketed at each step, convergence is . The method is first order and is exact for linear *f*.

2.4.3 Newton-Raphson Method

Consider the Taylor Series expansion of $f(x)$ about some point $x = x_0$:

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \tfrac{1}{2}(x-x_0)^2 f''(x_0) + O(|x-x_0|^3).$$

Setting the quadratic and higher terms to zero and solving the linear approximation of $f(x) = 0$ for x gives $x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$ and subsequent iterations are defined in a similar manner as $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

Geometrically, x_{n+1} can be interpreted as the value of x at which a line, passing through the point $(x_n, f(x_n))$ and tangent to the curve $f(x)$ at that point, crosses the y axis.

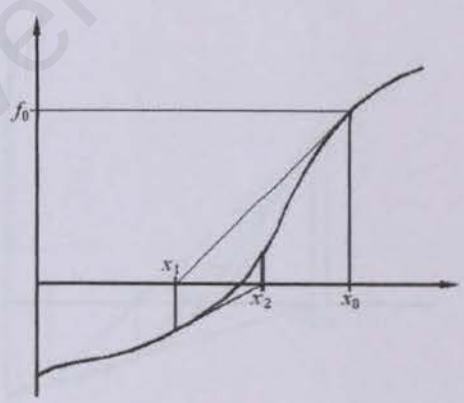


Figure 2.10: Newton-Raphson Model

2.4.4 Secant Method

This method is essentially the same as Newton-Raphson [9] except that the derivative $f'(x)$ is approximated by a finite difference based on the current and the preceding estimate for the root. For example, the equation of $f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$ and this is substituted into the Newton-Raphson (8) to give $x_{n+1} = x_n - \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} f(x_n)$

This formula is identical for the Linear Interpolation method. The difference is that rather than replacing one of the two estimates so that the root is always bracketed, the oldest point is always discarded in favour of the new. This means it is not necessary to have two initial guesses bracketing the root, but on the other hand, convergence is not guaranteed.

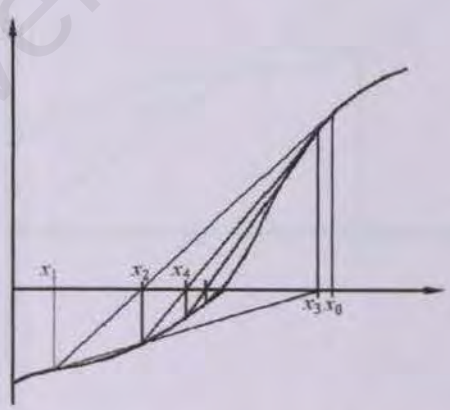


Figure 2.11: Secant Model

2.4.5 Direct Iteration

A simple and often useful method involves rearranging and possibly transforming the function $f(x)$ by $T(f(x),x)$ to obtain $g(x) = T(f(x),x)$. The only restriction on $T(f(x),x)$ is that solutions to $f(x) = 0$ have a one to one relationship with solutions to $g(x) = x$ for the roots being sort. Indeed, one reason for choosing such a transformation for an equation with multiple roots is to eliminate known roots and thus simplify the location of the remaining roots. The efficiency and convergence of this method depends on the final form of $g(x)$.

The iteration formula for this method is then just $x_{n+1} = g(x_n)$. A graphical interpretion of this formula is given the figure.

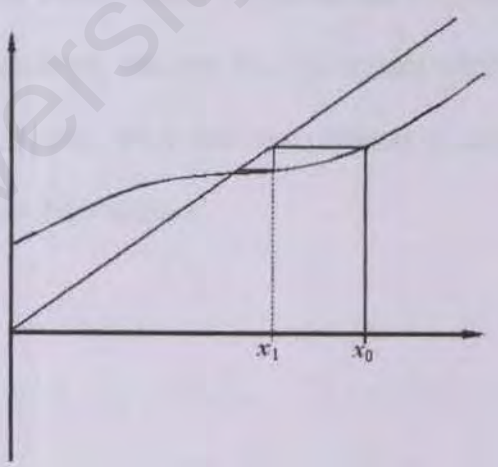


Figure 2.12: Direct Iteration Model

2.5 DISADVANTAGE OF SOME COMMON METHODS OF SOLVING TRANSCENDENTAL EQUATION

2.5.1 Analytical/Iterative methods to solve transcendental equations

Many analytical/iterative methods are used to solve transcendental equations [2]. Though these methods are capable of solving many transcendental equations they suffer from many common disadvantages. Usually transcendental equations have many solutions in a given range, and analytical methods are not able to find all these roots in a given interval, even when they find several solutions, it is not possible to conclude that the given method has found the complete set of roots/solutions, and has not missed any particular solution. Also, these methods fail in case of misbehaved or discontinuous functions. Hence, though these methods may work very well in some situations, they are not general in nature and need a lot of homework from the Analyst.

2.5.2 Disadvantage of Bisection Method

This method needs two points on the graph such that $f(a)*f(b)<0$. There is no straightforward analytical method to find these points [2]. Another problem lies in choosing the distance between the points a and b. For the method to work, a and b should be close enough, such that the function behaves monotonously in these limits. At the same time, a small difference in values of a and b makes it difficult to search the sample space. Further still, the method fails for discontinuities in function.

2.5.3 Disadvantage of Newton-Raphson Method

This is a commonly used method for solving transcendental equations. The method makes use of the slope of the curve at different points [2]. Therefore, if the function is non differentiable at points or has a point of inflexion, the method is not able to find the roots. Secondly, if the function changes its slope very quickly (frequently achieves slope of zero), or is discontinuous, the function cannot be solved by this method. If the function is discrete, the derivative has no meaning for it and this method cannot be used. Also there is no straightforward way to find all the roots in an interval or even ascertain the number of roots in the interval.

2.5.4 Disadvantage of False Position Method

This method suffers from same problems as the Bisection method. Hence it can be concluded that analytical methods cannot find all the roots of a transcendental equation reliably.

2.6 GENETIC ALGORITHM AS AN ALTERNATIVE METHOD OF SOLVING TRANSCENDENTAL EQUATION

2.6.1 GA As Optimization Problem

Solving transcendental equations is also a kind of optimization problem and hence GAs is applicable here [2]. The present problem has a large search space, a fitness function that might be misbehaved and has more than one solution. All these difficulties indicate that genetic algorithms may be useful in such situations.

2.6.2 GA Tackle Multimodal Problem

It is observed that the problem at hand is multimodal in nature i.e. one equation can have many roots [2]. The simple genetic algorithm is capable of searching optimum for a uni-modal function, but converges to some local optimum for a multimodal problem.

2.6.3 Implementation of GA of the Present Problem

2.6.3.1 Encoding

There is no strict rule for encoding the solutions to the problem. Generally binary coded solutions are used, though lately, real coded chromosomes are also being used.

2.6.3.2 Randomness

GAs relies in part on random sampling. This makes it a nondeterministic method, which may yield somewhat different solutions on different runs, even if the model is changed.

2.6.3.3 Population

Where most classical optimization methods maintain a single best solution found so far, an evolutionary algorithm maintains a population of candidate solutions. Only one or a few, with equivalent objectives of these is "best," but the other members of the population are "sample points" in other regions of the search space, where a better solution may later be found. The use of a population of solutions helps the evolutionary algorithm avoid becoming "trapped" at a local optimum.

2.6.3.4 Fitness function

The fitness function tells the algorithm how good a particular solution is. It is the objective function that the algorithm is supposed to be minimized.

2.6.3.5 Selection and reproduction operators

As the present problem is multimodal, the selection and reproduction operator depends on it. The method is kept free from differentiation or any other analytical methods to tackle non-differentiable and discontinuous functions.

2.7 MATLAB AS A TOOL FOR OPTIMIZATION PROBLEM

2.7.1 Introduction

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation [7].

Typical uses of MATLAB in engineering, mathematical and numerical analysis field include:

- Math and computation Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB features a family of add-on application-specific solutions called toolboxes that allow learning and applying specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems

2.7.2 Optimization Problem

Solving transcendental equations is also a kind of optimization problem and hence GAs is applicable here. MATLAB features a family of add-on application-specific solutions called toolboxes [7]. MATLAB provides Optimization Toolbox, which solve a variety of optimization problems, including a special section that highlights large-scale problems.

2.8 SUMMARY

Genetic algorithms are adaptive methods, which may be used to solve search and optimization problems, based on the genetic processes of biological organisms. The search starts from a population of many points, rather than starting from just one point. This parallelism means that the search will not become trapped on local maxima. The transition rules used by genetic algorithms are probabilistic, not deterministic. GAs makes the search domain transparent to the algorithm and frees it from the constraint of having to use auxiliary or derivative information. Solving Transcendental Equation using Genetic Algorithm entails that the researcher needs to find the solutions or roots for the equation using genetic algorithm. The problem of root finding, in a single variable can be thought of, as an optimization problem.

CHAPTER III

METHODOLOGY

CHAPTER III METHODOLOGY

3.1 WATERFALL SYSTEM DEVELOPMENT LIFE-CYCLE MODEL

The Waterfall Life-Cycle Model is used as a system development life-cycle model. This model is a linear life-cycle model with feedback loops, using documentation-driven concept. The advantages of the Waterfall Model, are, it is easy to understand and it emphasizes the documentation for an easy maintenance. The Waterfall Life-Cycle Model, which acts as a step-by-step guide, elicits the actual steps to perform during the project development, which makes it easy to follow and understand. This system development life-cycle model is suitable for the project as it is dedicated for small-scale projects [24].

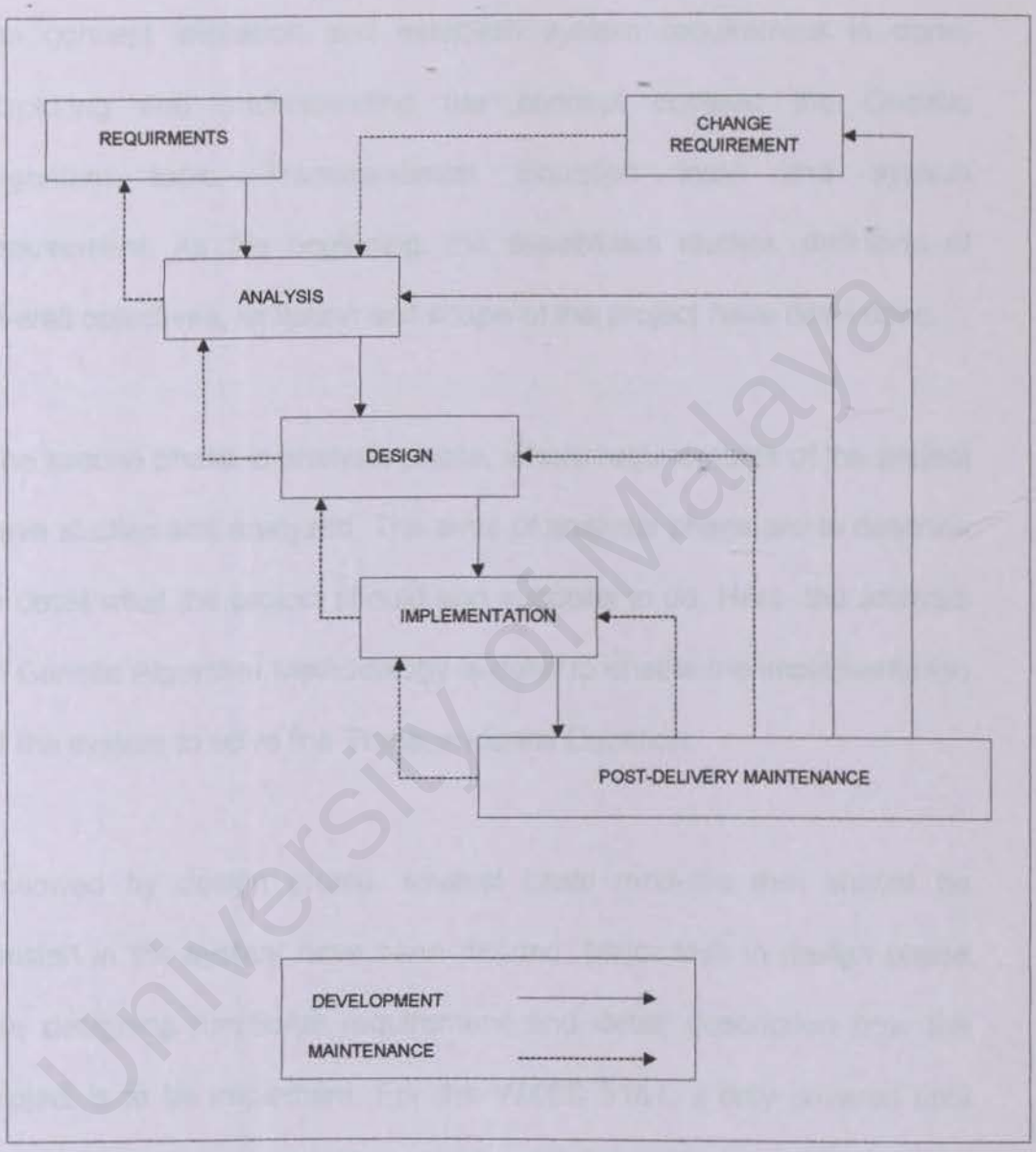


Figure 3.1: Waterfall System Development Life Cycle

3.1.1 Overview of Project Implementation

The implementation of this project begin with requirement phase, where the concept elicitation and establish system requirement is done. Exploring and understanding the concept covered the Genetic Algorithm topic, Transcendental Equation topic and system requirement. As the beginning, the feasibilities studies, definition of overall objectives, limitation and scope of the project have been done.

The second phase is analysis phase, where requirements of the project have studied and analyzed. The aims of analysis phase are to describe in detail what the project should and suppose to do. Here, the analysis of Genetic Algorithm Methodology is done to enable the implementation of the system to solve the Transcendental Equation.

Followed by design phase, several basic modules that should be existed in the system have been defined. Major task in design phase are designing functional requirement and detail description how the project is to be implement. For the WXES 3181, it only covered until design phase, while implementation and maintenance phase will be done during WXES 3182 in the next semester.

3.2 THE GENETIC ALGORITHM METHODOLOGY

Genetic Algorithm begins with the creation of a new population for the domain [23]. The Objective function is defined and here the objective is to find roots for the equation. The fitness function is derived from the objective function to determine the fitness for each candidate. Next, all the candidates are ranked based on their fitness value. The selection process is done step-by-step, followed by recombination and mutation process. The candidate will then be evaluated and will be reinserted into the new population. These looping processes will continue until we get a satisfactory result. This process is best illustrated by the flow chart as shown in Figure 3.2.

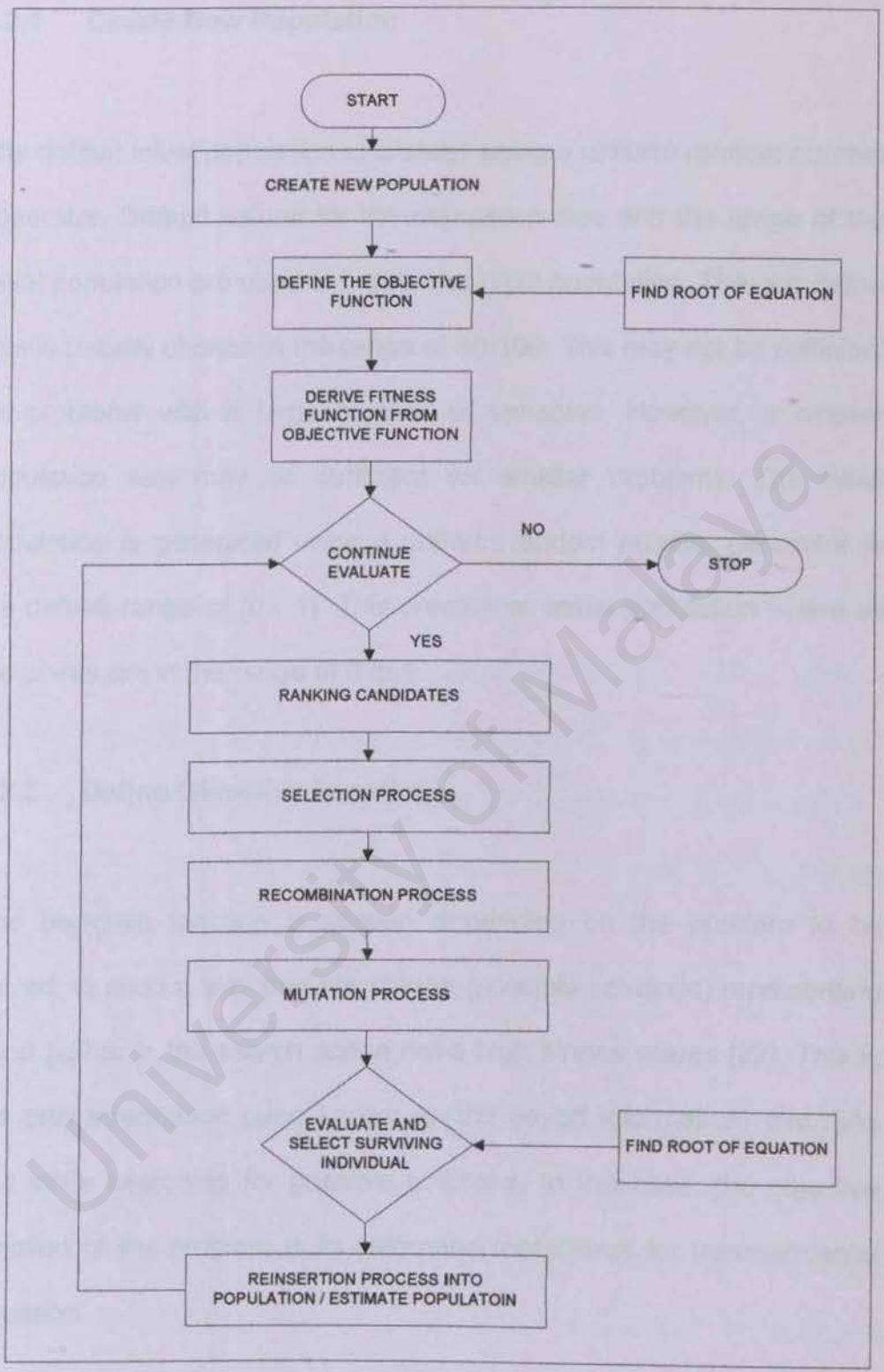


Figure 3.2: Genetic Algorithm Flow Chart For Solving Equation

3.2.1 Create New Population

The default initial population is created using a uniform random number generator. Default values for the population size and the range of the initial population are used to create the initial population. The population size is usually chosen in the range of 50-100. This may not be sufficient for problems with a large number of variables. However, a smaller population size may be sufficient for smaller problems. The initial population is generated using a uniform random number generator in the default range of [0 ; 1]. This creates an initial population where all the points are in the range of 0 to 1.

3.2.2 Define Objective Function

The objective function is chosen depending on the problem to be solved, in such a way that the strings (possible solutions) representing good points in the search space have high fitness values [22]. This is the only information (also known as the payoff information) that GAs use while searching for possible solutions. In this case, the objective function of the problem is to determine roots/zeros for transcendental equation.

3.2.3 Fitness Function

The fitness function is derived from the objective function by some application specific procedures [22]. For example, if the problem is that of minimization of some cost function (objective function), then the fitness has to be appropriately computed so that the fitness is inversely proportional to the objective.

3.2.4 Selection

The selection process copies individual strings (called parent chromosomes) into a tentative new population (known as mating pool) for genetic operations [22]. The number of copies that an individual receives for the next generation is usually taken to be directly proportional to its fitness value.

The standard selection techniques in genetic programming are Roulette Wheel and Tournament Selection. These techniques are by no means the only methods employed in genetic algorithm, which they are simply the most commonly employed. In this project, Stochastic Universal Selection and Roulette Wheel Selection will be used as the selection schemes. The primary task of all selection methods is to measure the relative fitness of each candidate.

3.2.5 Crossover

The main purpose of crossover is to exchange information between randomly selected parent chromosomes by recombining parts of their genetic material [3]. It combines parts of two parent chromosomes to produce offspring for the next generation. Single point crossover is one of the most commonly used schemes, and this scheme is used here. As a start, the members of the selected strings in the mating pool are paired at random. Then for each pair an integer position *k*, (known as the crossover point) is selected uniformly at random between 1 and l-1, where l>1 is the string length. Swapping all characters from position k+1 to l creates two new strings.

3.2.6 Mutation

Mutation is the process by which a random alteration in the genetic structure of a chromosome takes place. Its main aim is to introduce genetic diversity into the population [3]. Mutating a binary gene involves simple negation of the bit, while that for real coded genes are defined in a variety of ways. Here, mutation is done at 2 positions between the strings.

3.2.7 Set Up Parameters

There are several parameters in GAs that have to be manually tuned and fixed by the programmer. Some among these are the population size, string length, probabilities of performing crossover and mutation and the termination criteria [22]. Usually population size is kept fixed, probabilities of performing crossover are kept high and probabilities of performing mutation are kept low.

For most realistic cases, population size is usually chosen in the range 50-100, depend on the domain of the problem. Probabilities of performing crossover are usually chosen in the range of [0.6-0.9] and the probabilities of performing mutation are usually chosen in the range of [0.01-0.1]. For the string length, it usually depends on the required precision.

3.2.8 Stopping/Termination Criterion

The cycle of selection, crossover and mutation is repeated a number of times till one of the following occur [22]:

- The average fitness value of a population becomes more or less constant over a specified number of generations

- A desired objective function value is attained by at least one string in the population
- The number of generations (or iterations) is greater than some threshold

In this project, the criterion selected for termination was the third method, specified maximum number of generations. The main reasons to terminate a genetic programming run prior to the allowed generations are to reduce processor cost and develop a solution in the shortest period of time. However, additional termination criteria can be added like a desired objective function value is attained by at least one string in the population. This would have been likely influence the data produced through the experiment.

3.3 TRANSCENDENTAL EQUATION IMPLEMENTATION

The implementation of solving the transcendental equation briefly begins with the problem elicitation, followed by determining the search space and defining search space into interval. Using Genetic Algorithm as a solver, it constructs roots investigation in the interval. Detail information about the implementation will be discussed in the Chapter VI System Implementation.

CHAPTER IV

SYSTEM ANALYSIS

CHAPTER IV SYSTEM ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

The functional requirement describes the functionality of the program, as well as how the program should react to particular inputs and behave in a particular situation.

4.1.1 Setting Up a Problem for GA

The function of Genetic Algorithm searches for an unconstrained minimum of a function using the Genetic Algorithm. To use the Genetic Algorithm as a solver, we need to provide at least two input arguments: a fitness function and the number of variables in the problem [8].

The first two output arguments returned by the Genetic Algorithm are the best point found, and the function value at the best point. A third output argument will tell the reason why Genetic Algorithm stopped. Genetic Algorithm can also return a fourth argument (output), which contains information about the performance of the solver.

4.1.2 Specifying Population Options

The Initial population is created using a uniform random number generator. Default values for the population size and the range of the initial population are used to create the initial population [17].

4.1.2.1 Population Size

The default population size relies on the population size and number of variables. Large population size may be suitable for problems with a large number of variables, or a smaller population size may be sufficient for smaller problems.

4.1.2.2 Population Range

The initial population is generated using a uniform random number generator in a default range of [0 ;1]. This creates an initial population where all the points are in the range 0 to 1. For example, a population of size 3 in a problem with two variables could look like:

Population = rand(3,2)

Population =

0.1026	*0.2810*
0.8642	*0.7818*
0.1511	*0.0476*

4.1.3 Choosing Genetic Algorithm Operator

Genetic Algorithm starts with a random set of points in the population and uses operators to produce the next generation of the population. The different operators are scaling, selection, crossover, and mutation. There are several functions to choose from for each operator. The best function value may improve or it may get worse by choosing different operators [18].

4.1.4 Modifying Stopping Criteria

As described in Chapter 3, the Genetic Algorithm uses four different criteria to determine when to stop the solver. The Genetic Algorithm stops when the maximum number of generations is reached. Genetic Algorithm also detects if there is no change in the best fitness value for some time given in seconds (stall time limit), or for some number of generations (stall generation limit). Another criterion is the maximum time limit in seconds [18].

4.1.5 Reproducing Results

By default, the Genetic Algorithm starts with a random initial population which is created using MATLAB random number generators. The next generation is produced using Genetic Algorithm

operators that use these same random number generators. Every time a random number is generated, the state of the random number generators change. This means that even if there is no change in the random number used, when it runs again, the generator will produces different results. The results will be different because the states of the random number generators have changed from one run to another [18].

We can reproduce our results if we reset the states of the random number generators by using the information returned by the Genetic Algorithm. The Genetic Algorithm returns the states of the random number generators in the output argument. This information can be used to reset the states so the results of the next two runs are the same.

4.1.6 Visualization and Monitoring Performance

The Genetic Algorithm architecture is design to accept one or more plot functions through an options argument. This feature is useful for visualizing and monitoring the performance of the solver at run time.Options argument creates an options structure in order to select two plot functions. The first plot functions plots the best and mean score of the population at every generation. The second plot function plots the percentage of the stopping criteria that is satisfied.

4.2 NON-FUNCTIONAL REQUIREMENT

Nonfunctional requirement is a description of other features, characteristics and constraint that define a satisfactory system.

4.2.1 Friendly User Interface

The design of the Graphical user interface (GUI) must be attractive and the interface should be easy to use and understand. The arrangement of the menus must be systematic, so that users can easily understand the main purpose of the program.

4.2.2 Efficiency

The system should be implemented in the most efficient manner with the optimal access and producing the result with optimal accuracy. The efficiency of the program is determined by the right selection of population option, genetic algorithm operators and stopping criteria.

4.2.3 Reliability

In the case of this project, reliability means the results of the program must be reliable, the implementation of genetic algorithm to the domain produce a satisfactory solution. The solution must have strong reason why it behave or response, to be an optimal solution within a certain limit.

4.2.4 Maintainability

This application is design so that the efforts required maintaining, locating and fixing the errors in the system is as minimum as possible. It is done by well structured of the coding with clear explanation with a systematic data flow and interface, so that modification after the implementation is easier. The application could also be adapted to allow enhancement in the future. Documentations are provided to ensure the application is easy to maintain.

4.3 SOFTWARE REQUIREMENTS

MATLAB is a collection of functions that extend the capability of the numeric-computing environment [7]. It allows generating for many types of optimization including:

- Unconstrained nonlinear minimization
- Constrained nonlinear minimization
- Quadratic and linear programming
- Nonlinear least squares and curve-fitting
- Nonlinear system of equation solving
- Constrained linear least squares
- Sparse and structured large-scale minimization

4.4 HARDWARE REQUIREMENTS

4.4.1 Platform Specific Requirements

Operating System	Processor	Disk Space	RAM
Windows XP Windows 2000 Windows NT	Pentium III, Pentium IV, Pentium M, AMD Athlon, Athlon XP, Athlon MP	400MB (MATLAB ONLY with Help)	256MB 512MB (recommended)

Table 4.1: Requirement Specification

4.4.2 Other Recommended Items

- MS Windows supported graphics accelerator card
- MS Windows supported printer
- MS Windows supported sound card
- Office 2000 or Office XP is required to run MATLAB Notebook, MATLAB Excel Builder, Excel Link, Database Toolbox and/or MATLAB Web Server.

CHAPTER V

SYSTEM DESIGN

CHAPTER V SYSTEM DESIGN

5.1 INTRODUCTION

In the previous chapter of system analysis, we identified the requirements of the system, in other words, what is needed in order to build. In the system design phase, how the proposed system will be built is determined. This starts with designing the algorithm, followed by the data flow and the user interface design.

5.2 DESIGNING THE ALGORITHM

5.2.1 Setting Up Problem For GA

As has been discussed in the previous chapters, the genetic algorithm development begins with creating the initial population and determining the objective function/fitness function evaluation. Using a fitness function and the number of variables in the problem will generate the best solution or the function value at the best point. The stopping criteria will indicate when and why the GA should stop. From that point, an output can be generated, which may contain the information about the performance of the GA as a solver [1].

5.2.2 Visualization and Monitoring Performance

The visualization and monitoring performance tool is useful for visualizing the performance of the solver at run time. There will be two plot function; the first, will show the best and mean score of the population at every generation while the second plot function will plot the percentage of the stopping criterion satisfied [2].

5.3 DATA FLOW

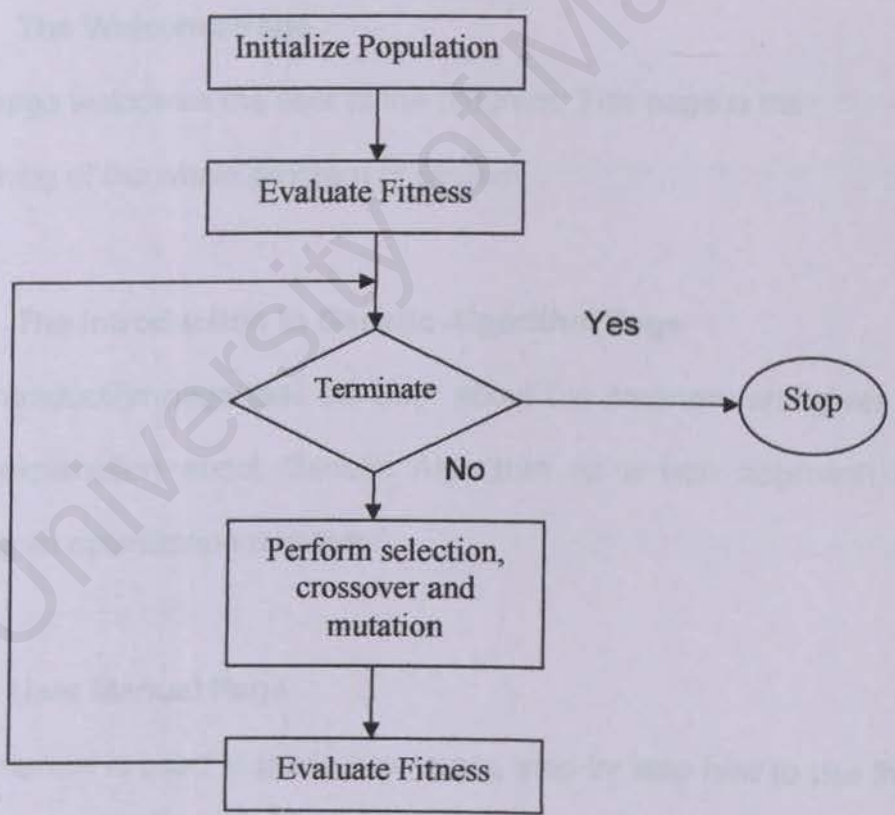


Figure 5.1: Designing Genetic Algorithm

The data flow of the program begins with population initialization. Then, the fitness function or objective function is evaluated for each individual in the population. The Looping process of selection, crossover and mutation is done until the evaluation of fitness has found the optimal solution or fulfill the stopping criterion. Then, the process results in the solution to the problem.

5.4 THE USER INTERFACE (GUI)

5.4.1 The Welcome Page

This page welcomes the user to the program. This page is the beginning of the whole program or system.

5.4.2 The Introduction to Genetic Algorithm Page

The introduction page tells the user about the program, and gives a brief explanation about Genetic Algorithm as a new approach in solving an optimization problem.

5.4.3 User Manual Page

User manual is used to guide new users, step by step how to use the application in correct manner in order to get the optimal solution.

5.4.4 Main System Page

This page is the main projector for the program where user selects the methods and several other options for Genetic Algorithm properties in order to solve the equation.

5.4.5 Output Page

In this page, the progress information about solver at run time will be available. Thus, the user is able to use the data or information for further observation or understanding.

5.4.6 Monitoring Performance

The monitoring performance is useful for visualizing the performance of the solver at run time. It will show the plotting graph for the desired result.

5.4.7 Information Page

It consists of detail explanation about jargon/function used in the application. This explanation will help new users to understand what and how the program work.

CHAPTER VI

SYSTEM IMLEMENTATION

CHAPTER VI SYSTEM IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the process of translating the detailed design into codes. By integrating the codes and interfaces, this will make up the desired system. Requirement analysis, design and implementation phases do not have a clear boundary in software development. Each phase tends to overlapped with each other. Implementation is based on the results of the system design.

6.2 SYSTEM IMPLEMENTATION PLAN

In the implementation phase, the designed modules are coded and modules compiled to make a complete model. The system architecture is captured during the design phases and may be added or modified during the code evaluation phase. Implementation includes:

- Implementing the designed classes and subsystems. In particular, the designed classes are implemented as file components that contain the source code.
- Coding all the modules into meaningful codes
- Using the modules integration

- Planning the testing strategy including unit testing, module testing, and component testing before integrating them into a system.
- Testing iteratively through all the process of development by using dummy data before using the real data.
- Building the interface and integrate them with the codes
- Testing the functionality of the interface after integrated with the system

6.3 SYSTEM DEVELOPMENT

System development begins with coding the designed classes, subsystems and modules. Each component is designed as simple as possible, free of error and created with understandable attributes. There are several modules that make up the Genetic Algorithm process. The modules are Fitness Function Module, Population Module, Genetic Algorithm Properties Module, Genetic Algorithm (GA) Solver Module, Output Module and the Graphical User Interface (GUI).

6.3.1 Fitness Function Module

The Fitness Function Module is the objective function defined by solving the equation to get it boundary and form it into a simpler and readable

form in the MATLAB environment. The range of the boundary is very important because it indicate the individual population creation, in order to obtain the optimal solution at the desired domain. This part is a core module, which is that the solved equation must be readable by both the MATLAB and also the Genetic Algorithm (GA) solver.

6.3.2 Population Module

Population modules are the population initialization based on the population initial range and the population size that is set up either by the programmer or the user. The module then generate random individual in the range of the boundary, which is here is set up [-1 1] up to the number of population size.

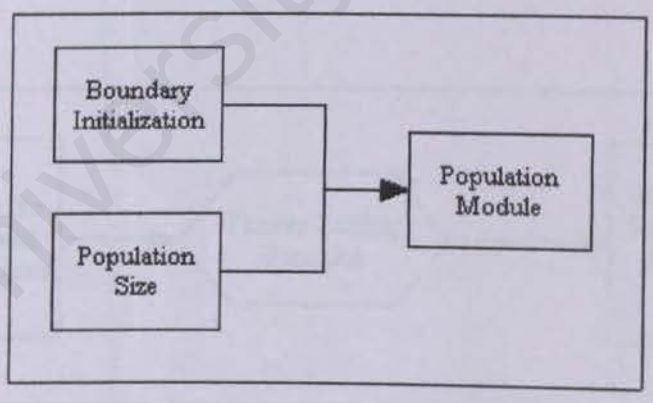


Figure 6.1: Population Module

6.3.3 Genetic Algorithm (GA) Properties Module

Genetic Algorithm Properties consists of several sub-modules, such as Fitness Scaling Function, Selection Function, Reproduction Function, Crossover Function, Mutation Function, Migration Function and Stopping Function.

6.3.3.1 Fitness Scaling Function

This function converts the raw fitness value returned by the fitness function to a value in a range that is suitable for the Selection Function. In this project ranking method is used, and raw scores are based on the rank of each individual. The rank of an individual is its position in the sorted scores.

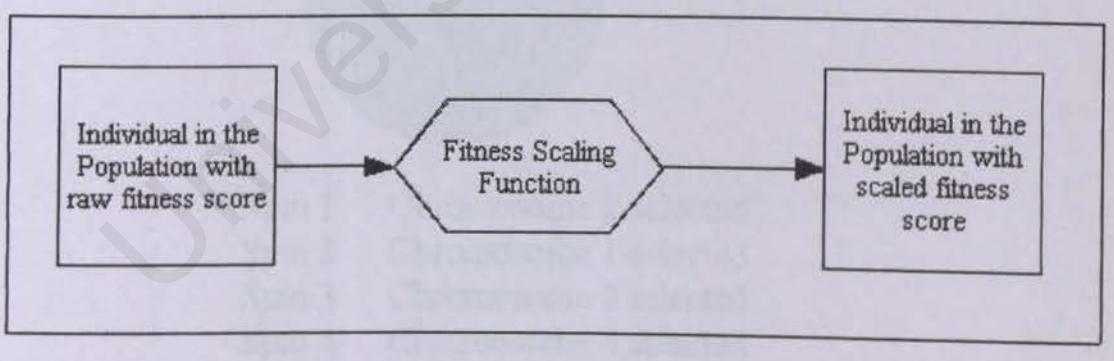
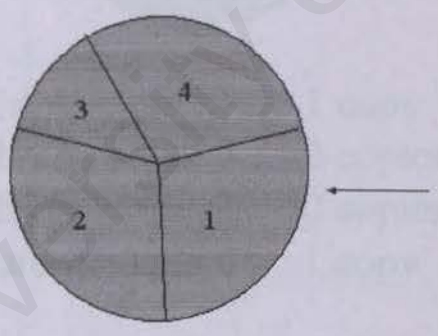


Figure 6.2: Fitness Scaling Function

6.3.3.2 Selection Function

The Selection Function chooses parents for the next generation based on their scaled values from the Fitness Scaling Function. For this project, two methods used namely the Roulette Selection and the Stochastic Uniform Selection.

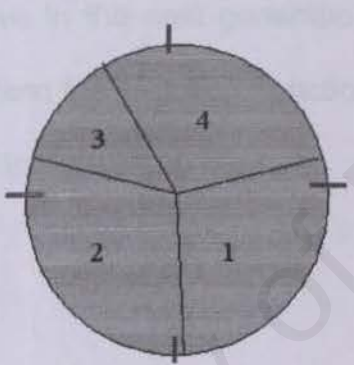
- The Roulette Selection simulates a roulette wheel I which the area of each segment is proportional to its expectation. The algorithm then uses a random number to select one of the sections with a probability equal to its area.



Spin 1 Chromosome 2 selected
Spin 2 Chromosome 1 selected
Spin 3 Chromosome 2 selected
Spin 4 Chromosome 4 selected

Figure 2.3: Roulette Wheel Selection Model

- The Stochastic Uniform Selection lays out a line in which each parent corresponds to a section of the line of length proportional to its expectation. The algorithm moves along the line in steps of equal size, one step for each parent. At each step, the algorithm allocates a parent from the sections it lands on. The first step is a uniform random number less than the steps size.



Chromosome 1	1 copy
Chromosome 2	2 copies
Chromosome 3	0 copies
Chromosome 4	1 copy

Figure 2.4: Stochastic Universal Selection Model

6.3.4.3 Creation Function

This function determines how the Genetic Algorithm (GA) creates initial children at the beginning of the process. This module generates randomly individuals in the first generation. Population Size indicates numbers of individuals. For the creation for the next generation, children are produce by the Crossover Function and the Mutation Function. It also consists of the Elite Count, which are specifies the number of individuals that are guaranteed to survive in the next generation without modification of the Crossover Function and the Mutation Function. The number of Elite Count is less than or equal to Population Size.

6.3.4.4 Crossover Function

The Crossover Function combines two individuals or parents to form a new individual or child for the next generation. Here, the implementation had been conducted to try three different methods namely the Single Point, the Two Points and the Scattered.

- The Single Point method selects a vector entry randomly, and both selected parents swap each other to form a child.

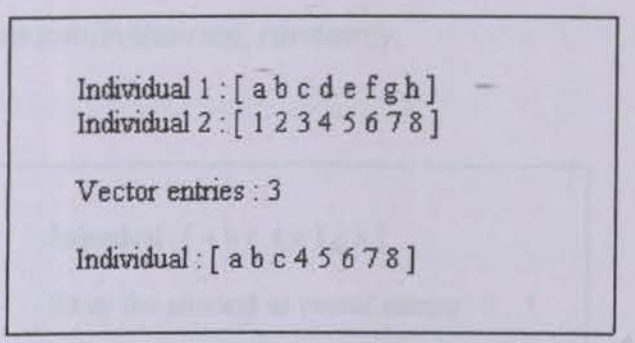


Figure 6.3: Single Point Crossover

- Two Points method selects two vector entries randomly, and both selected parents swap each other to form a child.

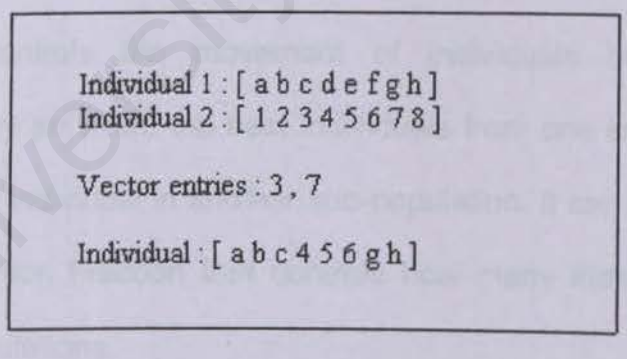


Figure 6.4: Two Points Crossover

6.3.4.5 The Mutation Function

The Mutation Function swaps the elements in the individual through vector entries based on mutation the rate, randomly.

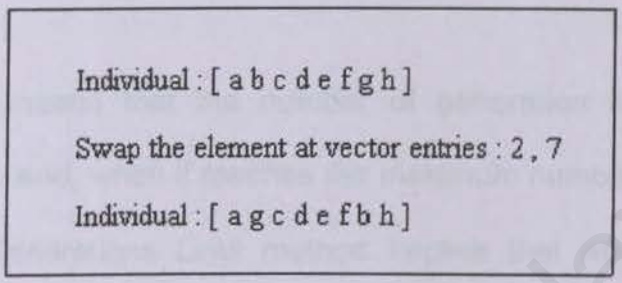


Figure 6.5: Mutation Diagram

6.3.4.6 The Migration Function

This function controls the movement of individuals between sub-populations. Every so often, the best individuals from one sub-population replace the worst individual in another sub-population. It can be control by setting the Migration Fraction that controls how many individuals move between sub-populations.

6.3.4.7 The Stopping Function

The Stopping Function indicates when the algorithm should stop. Here, the implementation consists of two methods, namely are the Generation Method and the Stall Generation Limit method.

- Generation means that the number of generation will indicate the simulation to end, when it reaches the maximum number of generation.
- The Stall Generations Limit method implies that when there is no improvement in the fitness function value, the simulation should stop within a certain range of generation, without reaching the maximum number of generation.

6.3.4 Genetic Algorithm (GA) Solver Module

The GA Solver is the main function of the algorithm, and it is the projector of the simulation. The GA Solver integrates the Fitness Function Module, the Population Module and the GA Properties Module.

- Firstly, the GA Solver should be able to read the Fitness Function Module and the interpretation of the objective function.

- Next, the GA Solver must be able to integrate the Fitness Function Module with the GA Properties, in order to obtain the optimal result for the optimization process. When the integration is carried out properly, then the algorithm will executes and gives the result of the simulation as desired.

- The results may return a different value from the previous run, as the creation function for the initial Population is done randomly. However, the result is still the optimal solution for the domain.

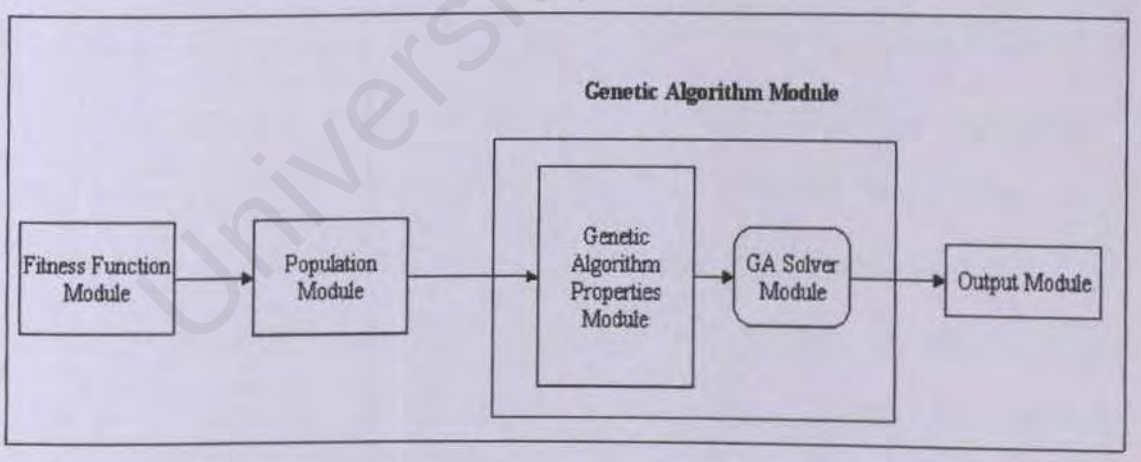


Figure 6.6: Genetic Algorithm Solver Module

6.3.5 Output Module

The Output Module returns the result of the simulation after the process has stopped. The result consist of the value for the best point found, the best function found, the reason why the GA Solver stop and the graph of the best individual found over the generation.

6.3.6 Graphical User Interface (GUI)

The graphical user interfaces consists of several pages namely the Welcome Page, the Introduction to GA Page, the User Manual Page, the Main System Page and the Output Page. The Output Page also consists of several other pages that plot the graph of the desired result. The flow of the graphical user interface (GUI) is as follows:

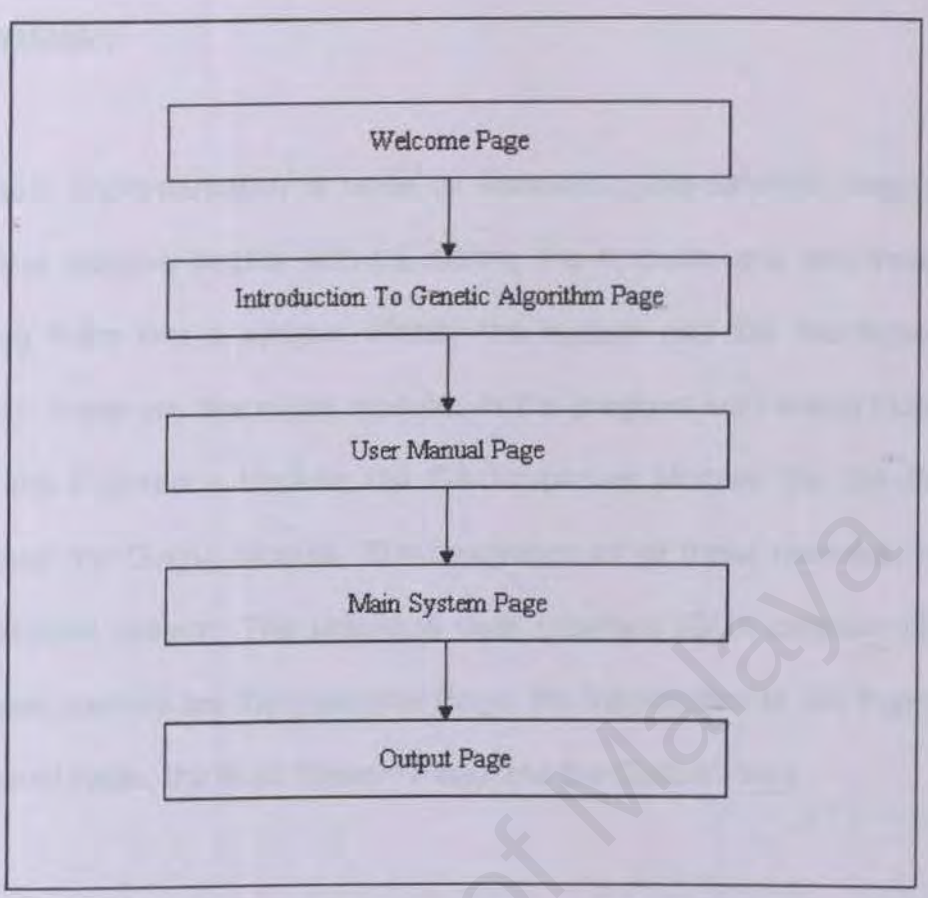


Figure 6.7: Graphical User Interfaces Flow Diagram

The graphical user interface (GUI) begins with the Welcome Page that leads to the next page, which is the Introduction to GA Page. This page gives a brief explanation about Genetic Algorithm in general. The User Manual Page that explains to new users how to use the program. In the Main System Page, all the GA properties are set up. After user has selected GA properties, the algorithm is executed and the solution appears in Output Section that is the Best Individual Found, the Best Function Value Found and the Stopping Condition. The plotted graph will appear in new pop-up window namely the Best Individual Found over the Generation.

6.4 SUMMARY

The system implementation is done by translating the detailed design into codes. The process begins with translating the modules and sub-modules, integrating them into a system. Finally the system and the interfaces are integrated. There are five major modules in the program are Fitness Function Module, the Population Module, the GA Properties Module, the GA Solver Module and the Output Module. The integration of all these modules, make up the desired system. The graphical user interface (GUI) consists of five main pages, namely are the Welcome Page, the Introduction to GA Page, the User Manual Page, the Main System Page and the Output Page.

CHAPTER VII

SYSTEM TESTING

CHAPTER VII SYSTEM TESTING

7.1 INTRODUCTION

Testing is the critical phase for the newly developed system. A new system must be tested in order to check that it meets with its objectives. No matter how a program is developed, it obviously suffers from a variety of errors and the modules should be checked to ensure that they function correctly.

7.2 TESTING PROCESS

The testing process is the market strategy to promote a reliable, user friendly and bug-free system. It contains some aspects that are oriented on the word of the system, which means that the testing should be done in the environment in which the program is being developed. The focus of the testing is component testing, integration testing and user testing.

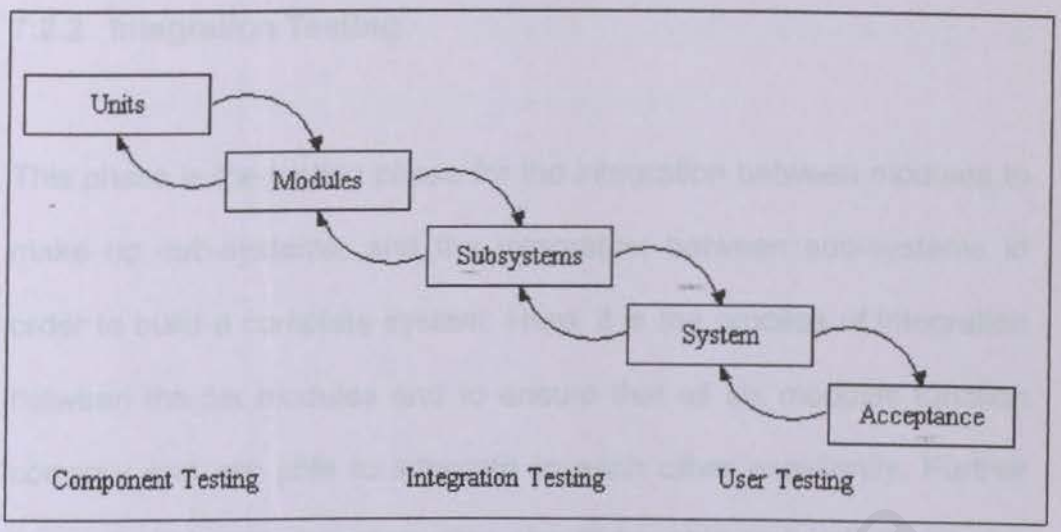


Figure 7.1: Testing Activities Flow Diagram

The figure depicts the whole process of the testing phase. It is important to understand that testing actually is done through out the system life cycle, only the activities are increased during the final phases of the System Development Life Cycle (SDLC). Component testing, integration testing and user testing is the sequence of testing activities. As defects are discovered at any stage, the program may be modified.

7.2.1 Component Testing

Component Testing consists of units and modules testing. The project consists of several modules and sub-modules. Each module has been tested to be free from bugs and syntax errors.

7.2.2 Integration Testing

This phase is the testing phase for the integration between modules to make up sub-systems, and the integration between sub-systems in order to build a complete system. Here, it is the process of integration between the six modules and to ensure that all six modules function correctly and are able to integrate to each other excellently. Further more, the integrated system also should be able to function with the Graphical User Interfaces (GUI) of the system.

7.3 TESTING STRATERGY

The Test Strategy is developed to detect and identify potential problems even the smallest errors in the system. The Test Strategy offers the road-map for the testing activities. The strategies used for the project are:

- Codes testing for syntax error and bugs.
- Integrated codes testing for functionality evaluation.
- Testing the integration between the system and the Graphical User Interfaces (GUI) using the dummy data, a simpler version of real data.

- Testing the integration between the system and the Graphical User Interfaces (GUI) with the real data.
- Test the whole system by testing the system and the interfaces using the real equation and real value of the equation.

7.4 SUMMARY

Testing activities consist of units, modules, sub-systems, system and acceptance testing phase. All these components are tested in sequence beginning with the Component Testing, followed by the Integration Testing and ending with the User Testing. All these activities are carried out in order to detect defects and errors, so as the system can be modified in order to ensure that the whole program functions correctly and successfully.

CHAPTER VIII

RESULT

CHAPTER VIII RESULT

8.1 INTRODUCTION

A result of this project will be discussed in this chapter. It covers the results, system's strengths and limitation. A few suggestions will be made as enhancement of the system in the future

8.2 RESULTS

As the result of the program, the program is able to find set of optimal solution that satisfied the equation, Y = 5 COS (X). Objective of this project is to find the minimum of the equation, and this returned the set of optimal solution.

The set of optimal solution is namely as the Best Individual Found and the Best Fitness Value Found. The Best Individual Found and the Best Fitness Value Found are obtained after the simulation is executed over the generation.

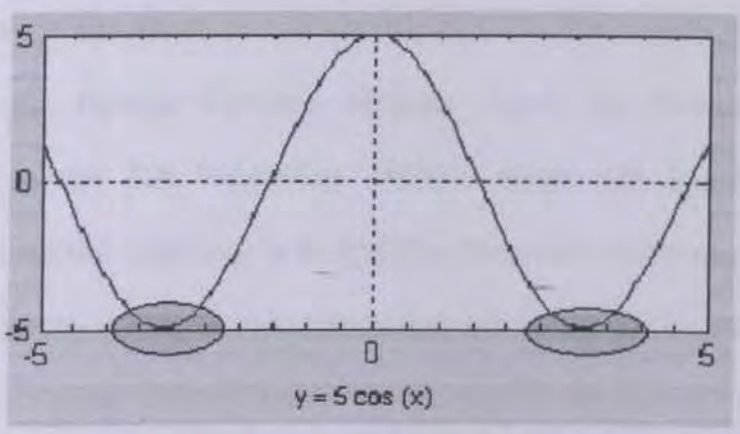


Figure 8.1: Graph of Y = 5 COS (X)

Concept of the Genetic Algorithm is to find optimal solution in the problem. In this project, it is intending to find the minimum of the equation Y = 5 COS (X). From the graph, the minimum value of the equation is Y = (-5). Along the X-axis, there are many points that returned Y = (-5), but in the range [-5 ; 5] there are only two points that returned Y = (-5). Genetic Algorithm is applied here not only to find the two values of (X) that return the Y = (-5), but to find a set of value or solution that returned the value of Y approximately Y = (-5). For example, Y = (-4.99879).

The initial range that is set up in the program is [0 ; 1], which means that the algorithm generates randomly the value of (X) or individuals depending on the Population Size, for example 20 in the range of [0 ; 1]. This is done by the Creation Function in the GA Properties Module and the Population Module. Then, the algorithm calculates the Fitness

Value of each individual, or namely the Score of the individual, which is done by the Fitness Function Module. Then, the Fitness Scaling Function in the GA Properties Module ranks the scores of the individual. As the objective is to find the minimum of the equation, the minimum scores in the current population will be the in the first rank.

The creation of the individuals or namely the Children for the next generation is produced by the Crossover Function and the Mutation Function. The selection of individuals for the creation of children, or namely the Parents is done in the Selection Function. Two methods applied are the Roulette Wheel Selection and the Stochastic Uniform Selection.

As the initial range is set up to [0 ; 1], the individual evolve to the right and the left of the graph towards the solution, Y = (-5). The solver stopped when the stopping condition is met, either the Generation or the Stall Generation Limit.

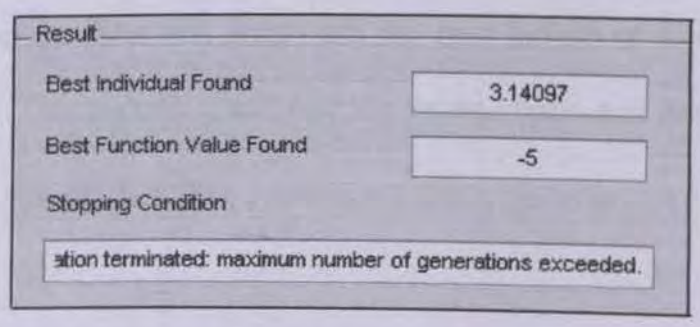


Figure 8.2: Result of the Program

8.2.1 Best Individual Found

The individual is actually any value of (X) that satisfied the equation, or in this project is the Y = 5 COS (X). However, the Best Individual Found is the best individual that satisfied the equation over the generation or before the stopping condition is met.

8.2.2 Best Function Value Found

The Best Function Value Found is the best scores calculated by the Fitness Function Module or we can say that is the value of Y that approximately Y = (-5) over the generation or before the stopping condition is met.

8.2.3 Graph

The result of the simulation is shown in the graph that plots two different graphs. The first graph plot the best individual found over the generation. The second graph plot the worst, mean and best individual found over the generation.

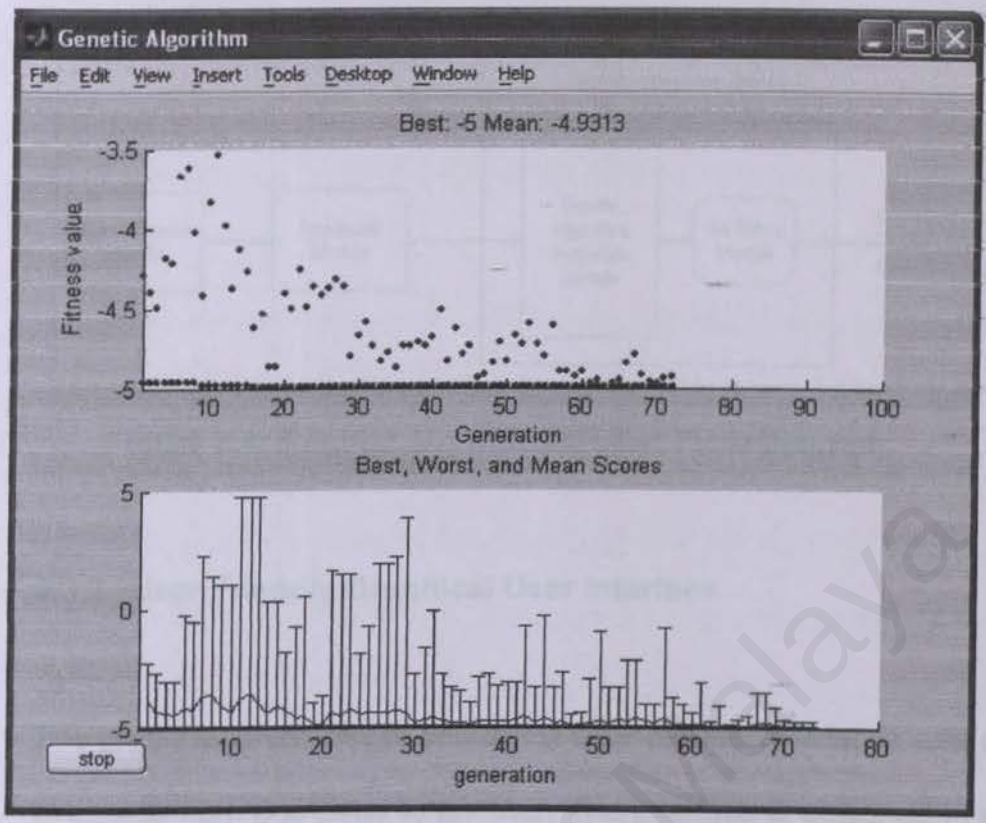


Figure 8.3: Plotted Graph of the Result over the Generation

8.3 SYSTEM STRENGTH

8.3.1 Understandable Program Flow

The program incorporates understandable program flow that is easy to understand. As result, the program is easy to maintain and modifications are possible for future enhancement. It incorporates five clearly expressed modules with several sub-modules, which are integrated in a sequence process.

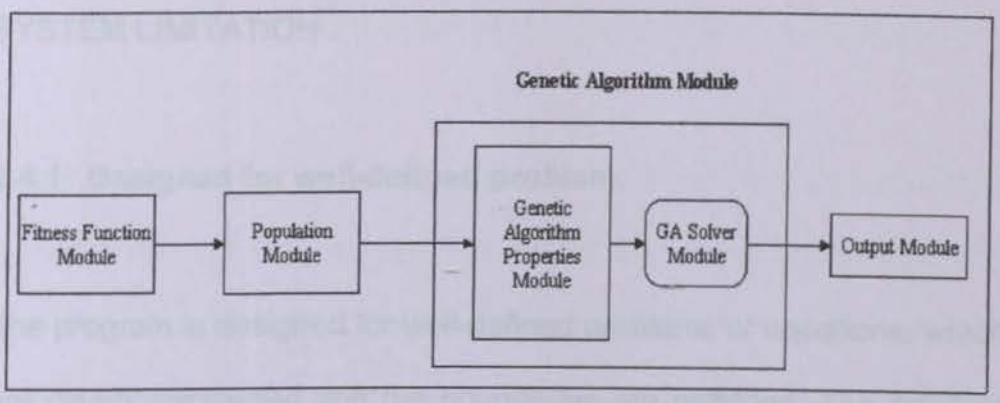


Figure 8.4: Genetic Algorithm Solver Flow Diagram

8.3.2 User Friendly Graphical User Interface

The program is created to present a user friendly interfaces and easy to access from one page to another. The interface is designed in such a way that is suitable and applicable to all level of user. It used direct and clear instruction.

8.3.3 Provide Several Options To User

In the Main System Page, the system provides several options in order to select several different techniques or methods for GA properties to solve the problems. Hence, users may choose the most suitable solution to solve the problem or equation and compare the result of each solution that fits best for them.

8.4 SYSTEM LIMITATION

8.4.1 Designed for well-defined problem

The program is designed for well-defined problems or equations, which are clearly expressed and the boundaries are provided. The problem or equation must be translated in form that is understandable by MATLAB as the system is developed in the MATLAB application. In this project, this program is concentrating in solving the equation, which is Y = 5 COS (X), so the result of the program is only to solve the equation.

8.4.2 Lack of Variety in GA Properties

As Genetic Algorithm techniques and methods are evolving through, there are many Genetic Algorithm technology but only small options of the Genetic Algorithm techniques is provided in the system. This system consists of only basic algorithm with only several options of methods especially the options in Genetic Algorithm properties. This is due to the lack of time and the skill of the developer.

8.4.3 Lack of Dynamic Aspect in Graphical User Interface (GUI)

The system is integrated with simple and easy access graphical user interface (GUI). The user interfaces not dynamic, since the system only solved the Y = 5 COS (X), in which the equation is embedded statically in the program. Used need to modify the codes if the users want to solved another similar equation.

8.5 FUTURE ENHANCEMENT

The recommendations for future work are to enhance some more features of Genetic Algorithm with the latest technology of GA, such as the Selection Module and the Crossover Module. The system should be modified so that the users are able to solve another similar equation not by modifying the codes, but through the interface. Further more, it is essential to design more dynamic user interface, corresponding to variety options for Genetic Algorithm, where user has more control to the system.

CHAPTER IX

DISCUSSION

CHAPTER IX DISCUSSION

9.1 INTRODUCTION

This chapter discuss about the problem faced during the implementation of the project Solving Transcendental Equation Using Genetic Algorithm. It consists of problem faced and solution and knowledge gain.

9.2 PROBLEM FACED AND SOLUTION

During the development and implementation phase, several problems are faced. During the development phase, the developer has encountered a problem in using and understanding the programming language used and the special functions of MATLAB. This problem is overcome by referring to books and MATLAB online help.

Next, as this is the first time ever I heard and learned about the Genetic Algorithm by myself, I confronted a problem to understand and construct the algorithm for the Genetic Algorithm. Not only that, I faced confusion to translate the algorithm into codes. To overcome the

obstacles, I have to refer existing codes in the internet, which had been written in different programming language to get the picture and idea about the algorithm, how its work and the codes.

During the implementation phase, after the algorithm has been translated into codes, the system was unable to read the problem. The algorithm cannot understand the Fitness Function Module, which caused the system failed to function correctly. Modification is made in the Fitness Function Module and coded back the equation to correct the error. Another problem during this phase is plotting the graph base on the output. This problem was overcome by referring to online help and existing demos in MATLAB.

9.3 KNOWLEDGE GAIN

At the beginning of the project, the researcher had very little knowledge about Genetic Algorithm, Transcendental Equation and MATLAB application. Through reading, surfing the internet, and consulting with her supervisor, this problem was overcome.

Transcendental Equation was learnt by surfing the net, asking friends from the Mathematic Department and reading some reference books. At the same time, the project supervisor was also consulted.

Meanwhile, MATLAB application was learnt through the internet and the online help and using trial and error techniques to explore the MATLAB. Friends who are familiar with the MATLAB environment was consulted in the creation of the Graphical User Interface (GUI) using MATLAB.

At the end of the project, the concept of Genetic Algorithm, which is a probabilistic method to solve optimization problem, but it is enhance with extra techniques and properties, was thoroughly understood. The MATLAB as the development environment was also understood.

Throughout this project, I learned that Genetic Algorithm provides solution for optimization problem such as solving the Transcendental Equation. It is able to find the optimum solution based on the problem.

REFERENCES

REFERENCE

[1] Madhukar, A. & Haraharan, R. (2002, January) A Genetic Algorithmic Approach To Find The Roots Of Equation: A Report. http://geocities.com/madhukar_anand02/rooteqn.pdf

[2] Varun Aggarwal. (2000) Solving Transcendental Equation Using Genetic Algorithms: A Technical Report. http://www.geocities.com/mumukshu/ste_gas.pdf

[3] Kumara, S. & David, E.G. (2004, January) Let's Get Ready To Rumble: Crossover Versus Mutation Head To Head: IlliGAL Report (No. 2004005) http://www-illigal.ge.uiuc.edu/~kumara/papers/2004005.pdf

[4] Yazann Romahi. (1998, February 12) Evolutionary Electronic. http://www.romahi.com/yazann/uni/diss.html

[5] Shirley C.B. (200, August 200). Pose Estimation For Soccer Robot Vision Using Genetic Algorithms: A Thesis Proposal. http://www.ccs.dlsu.edu.ph/Faculty/Caslon/pdf/pr0001a.pdf

[6] http://www.mathworks.com/products/gads/

[7] http://www.matlab.com

[8] http://www.mathworks.com/products/demos/gads/GA_options.html

[9] http://mpec.sc.mahidol.ac.th/numer/

[10] http://www.rennard.org/alife/english/gavintrgb.html

[11] http://www.uni_koblenz.de/~gb/projects/ale/studienarbeit_/html /node13.html

[12] http://www.uni_koblenz.de/~gb/projects/ale/studienarbeit_/html /node19.html

[13] http://www.goecities.com/mumukshu/gatrans/html

[14] http://www.egr.uh.edu/~sjorgens/Mga/genetic_toolbox/node4.html

[15] http://en.wikibooks.org/wiki/NumericM:Equation_Solving#Solution_of_ Algebraic_and_Transcendental_Equation

[16] http://www.damptp.cam.ac.uk/user/fdl/people/sd/lectures/nummeth98/ index.htm#L_1_Root_finding_in_one_dimension

[17] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/tcw2/article1.htm

[18] http://www.mathworks.com/products/gads/description3.html

[19] http://www.mathworks.com/products/gads/description6.html

[20] http://www.solver.com/gabasics.htm

[21] http://campus.northpark.edu/math/PreCalculus/Functions/index.html

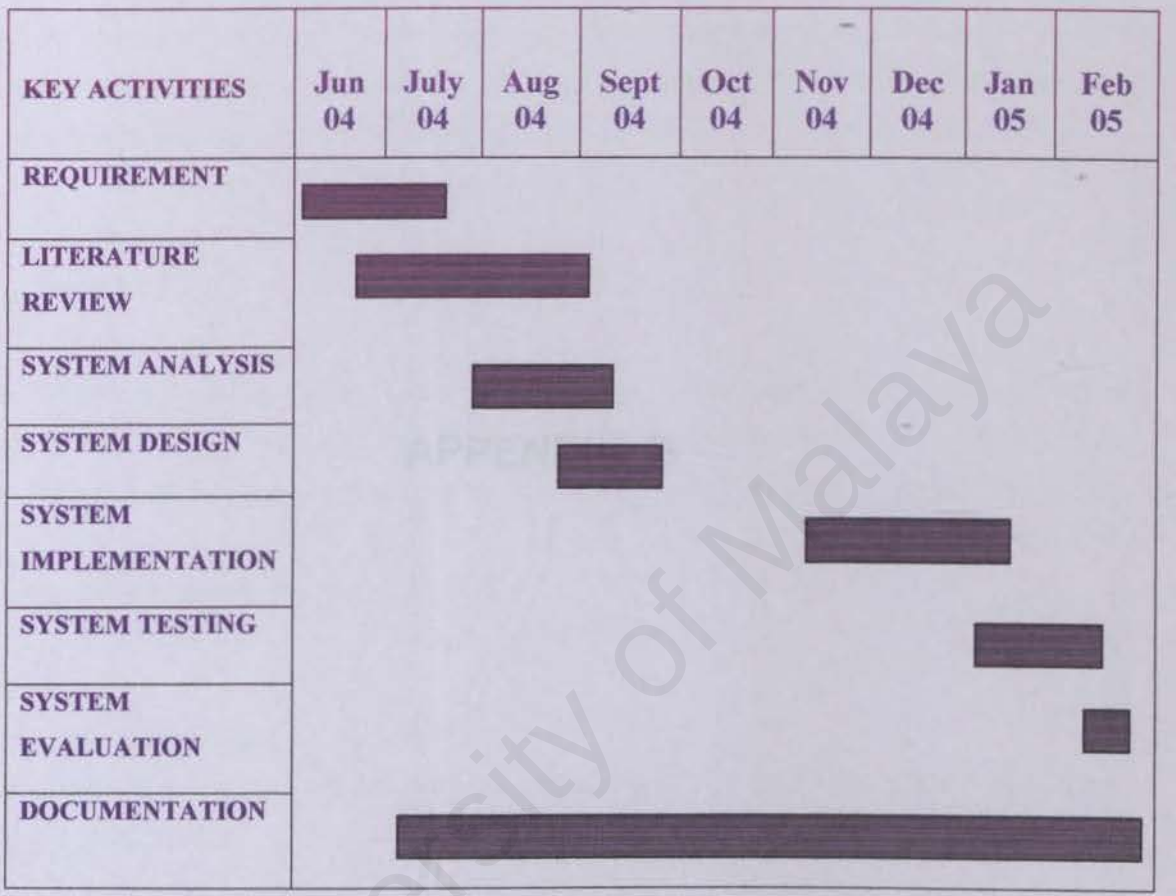
[22] http:// www.onlinemcle.com/brews/diss/chapter4.html

[23] Luger, G.F. (2002). Artificial Intelligence: Structures And Strategies For Complex Problems Solving (4th Ed.). London. Eddison Wesley.

[24] Schach, S.R. (2005). Object-Oriented And Classical Software Engineering (6th Ed.). London. Mc Graw Hill.

APPENDICES

APPENDIX A



KEY ACTIVITIES FOR THE PROJECT

APPENDIX B

Solving Transcendental Equation Using Genetic Algorithm System

This is the user manual for the Solving Transcendental Equation Using Genetic Algorithm System. This user manual will help the user and give instructions how the system works and how to handle the system.

Requirements

The system is build in the MATLAB 7 application. To run the system, your computer need to have all the requirements to run the system without errors. The requirements that need to install in your computer are:

General Requirements For MATLAB 7.0

- CD-ROM drive (for installation)
- Netscape Navigator 4.0 and above or Microsoft Internet Explorer 4.0 and above or Mozilla 1.x
- Adobe Acrobat Reader 3.0 or above (required to view and print the MATLAB online documentation in PDF format)
- Some license types require a license server running FLEXlm 9.2, which is provided by the MathWorks installer
- TCP/IP is required on all platforms when using a license server

Platform-Specific Requirements For MATLAB 7.0

Operating System	Processors	Disk Space	RAM
Windows XP 2000 (Service Pack 3 or 4) NT 4.0 (Service Pack 5, or 6a)	Pentium III, IV, Xeon, Pentium M, AMD Athlon, Athlon XP, Athlon MP	400MB (MATLAB ONLY with Help)	256MB 512MB (recommended)

For the computers those are not have the MATLAB 7.0, you need to install it first before you run the project. To run the application, your computer must at least has the minimum requirements mentioned above.

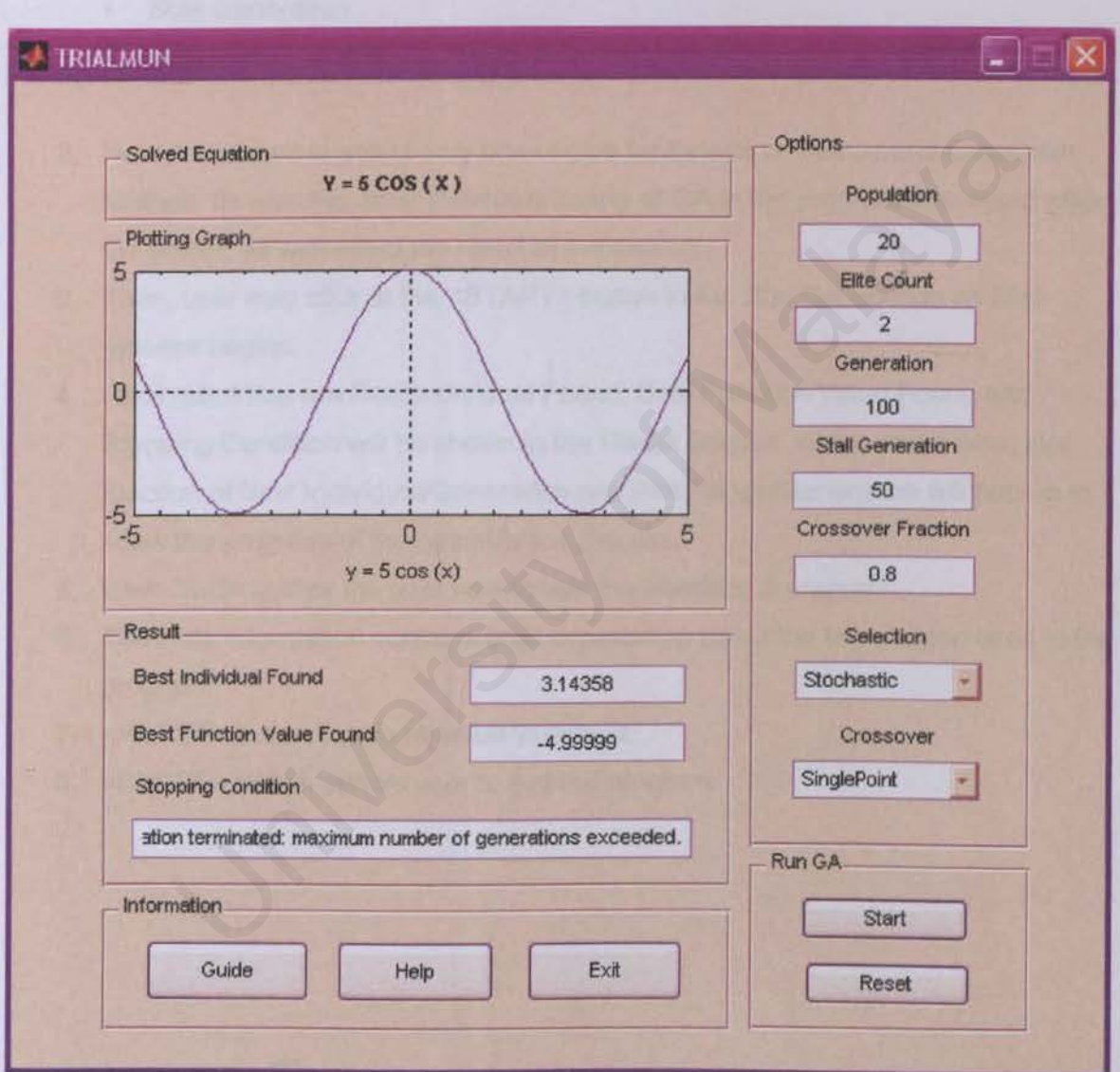
How to run the System

Manually:

1. Open the MATLAB application.
2. Then, open the file **Work → GA-Project → Interface**
3. Cilck at the Pg1.fig
4. Then ,it will generate the first page of the system and and the whole system is ready to use by the users.

How to Handle the System

The main page of the system is like the figure, where the user need to specify several properties/value in the page. As you can see, there are several sections in the interface. As to run GA, we need to follow the step-by-step to enable the GA to run. So, to use the program user should:

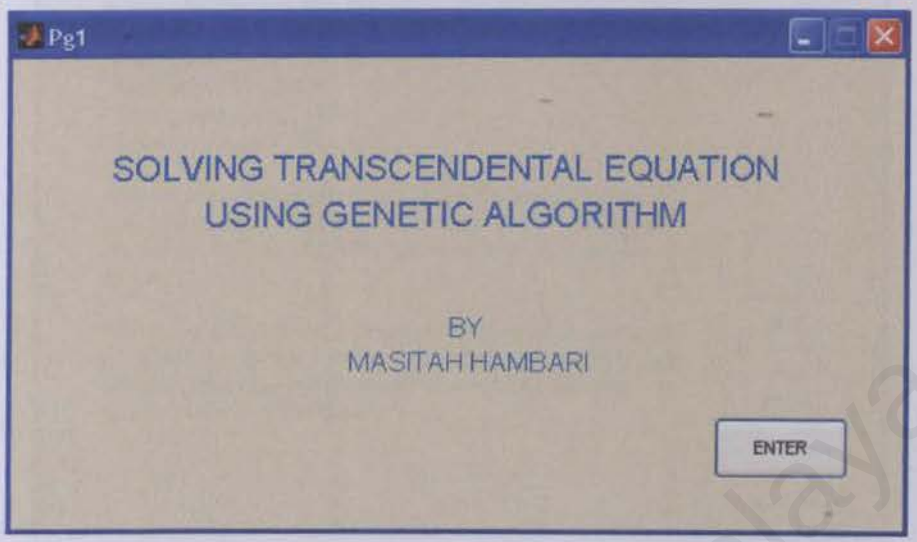


1. In the Options Section, there are several section that need user to insert certain value. All these variable are the internal property og the program that would effect the solver. The property are:

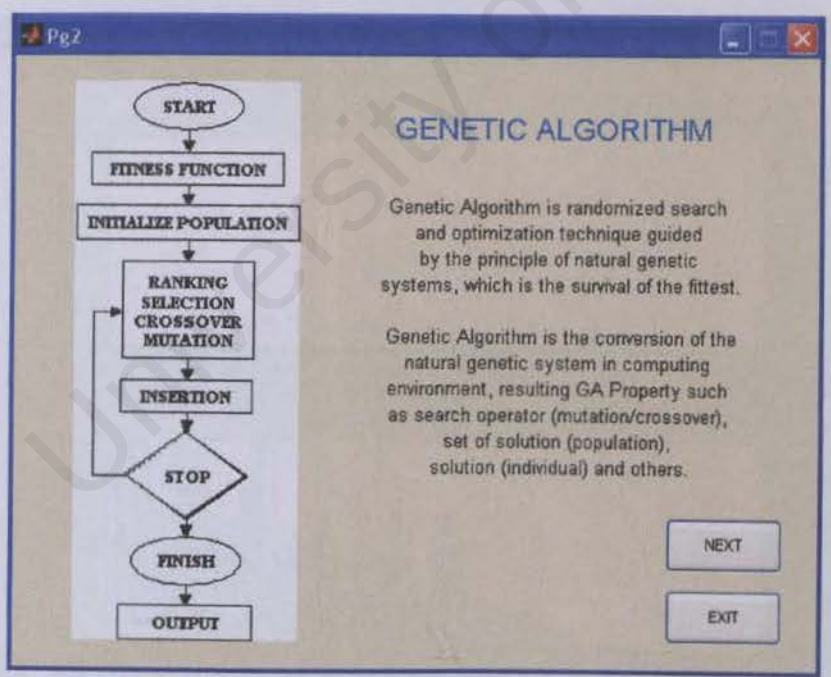
    - Population Size
    - Elite Count
    - Generation
    - Stall Generation
    - Crossover Fraction

2. Next, user need to select only one choice for Selection Method and Crossover Method. Its also the other internal property of GA in the program that would effect the solver, as well effect the result of the search.
3. Then, user may click at the <START> button in the Run GA Section and the process begins.
4. Final result that are Best Individual Found, Best Function Value Found and Stopping Condition will be shown in the Result Section. At the same time, plot function of Best Individual/Generation and Plot Range/Generation will pop-up to show the progress of the optimization process.
5. User Guide guides the user how to use the interface & program.
6. The Help Information contains brief explanation about the term/jargon used in the program.
7. <RESET> button return all value to default.
8. <CLOSE> button enable user to exit the program.

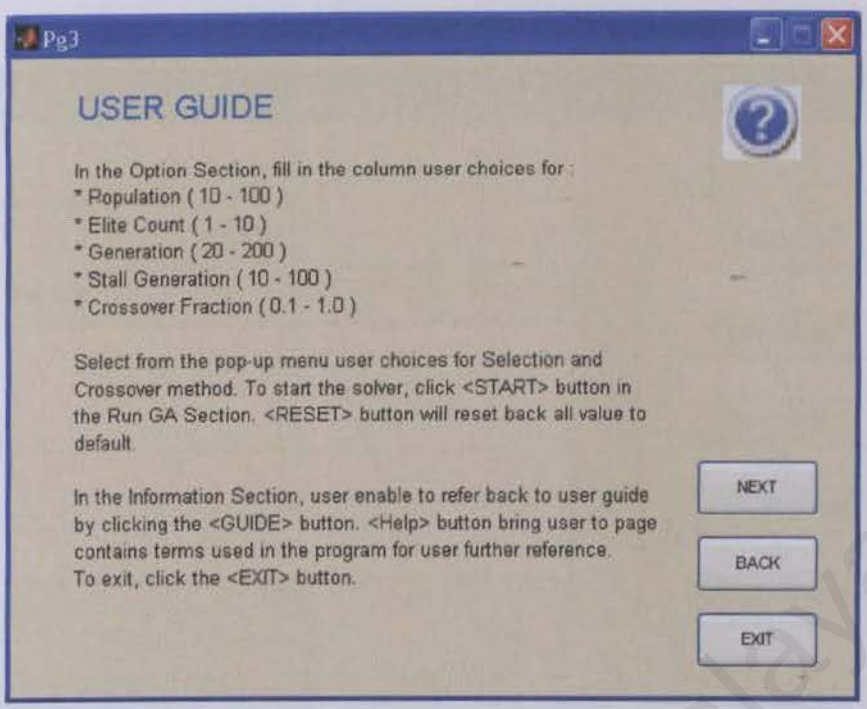
APPENDIX C



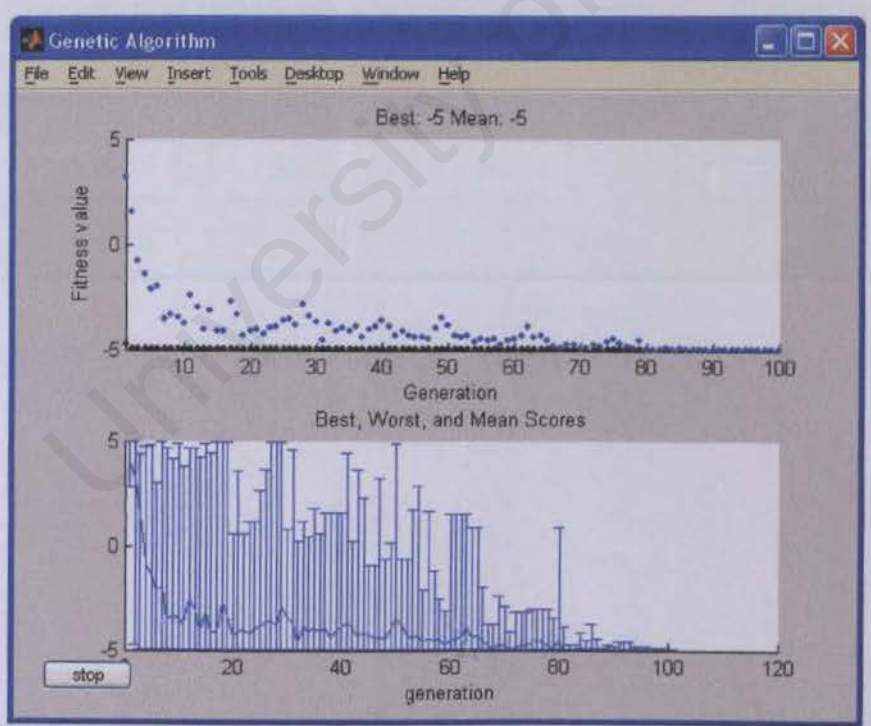
INTRODUCTION PAGE OF THE SYSTEM



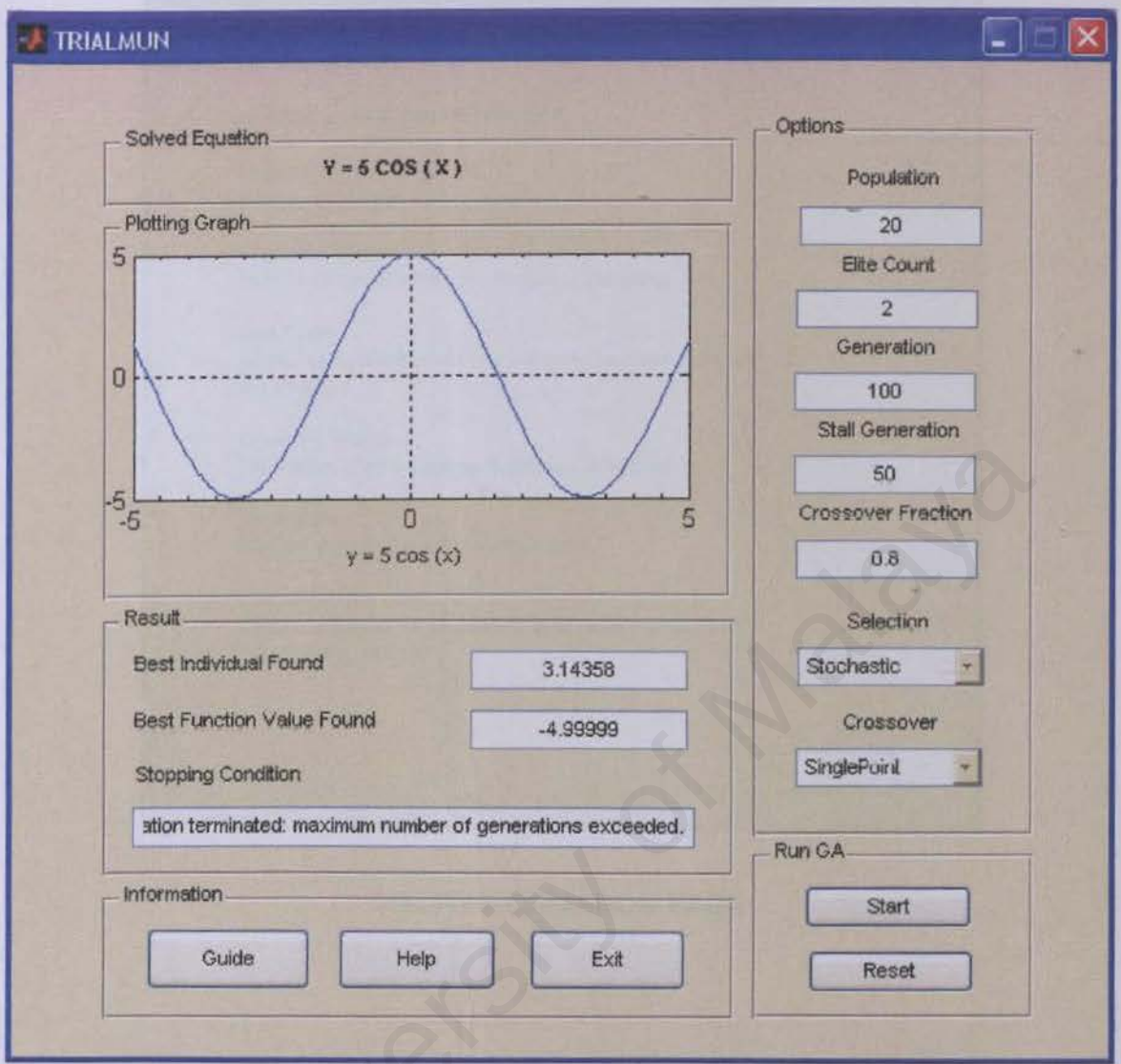
INTRODUCTION PAGE TO GENETIC ALGORITHM



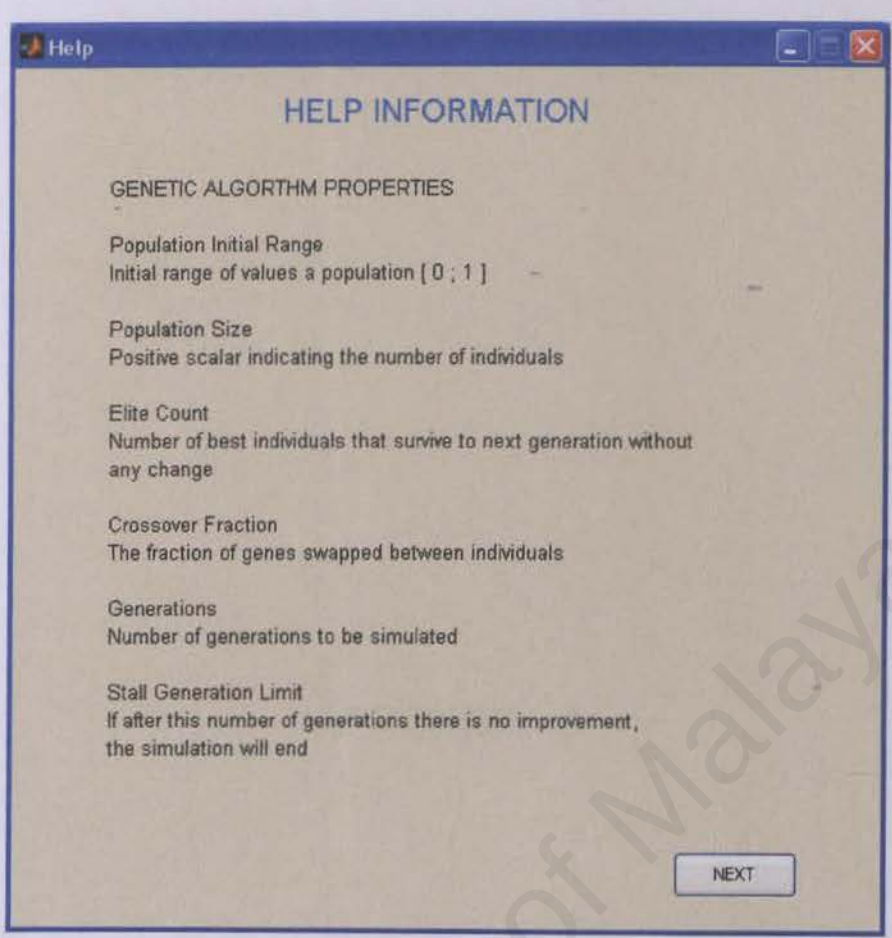
USER GUIDE PAGE



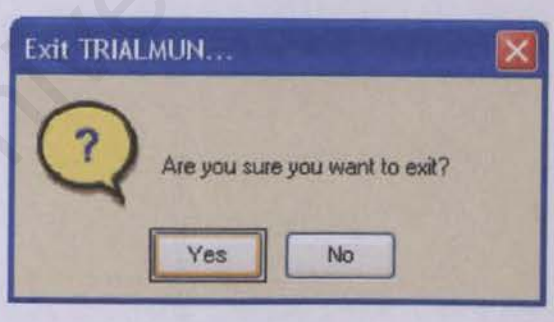
MONITORING PERFORMANCE PAGE



MAIN PAGE OF THE SYSTEM



HELP INFORMATION PAGE



WINDOWS PROMPT TO EXIT

APPENDIX D

M-File for the solved Equation

```
function y = simple(x)
y = 5*(cos(x));
```

M-File for the Main System

```
function varargout = TRIALMUN(varargin)

% TRIALMUN M-file for TRIALMUN.fig
%      TRIALMUN, by itself, creates a new TRIALMUN or raises the existing
%      singleton*.
%
%      H = TRIALMUN returns the handle to a new TRIALMUN or the handle to
%      the existing singleton*.
%
%      TRIALMUN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TRIALMUN.M with the given input arguments.
%
%      TRIALMUN('Property','Value',...) creates a new TRIALMUN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TRIALMUN_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TRIALMUN_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Copyright 2002-2003 The MathWorks, Inc.
% Edit the above text to modify the response to help TRIALMUN
% Last Modified by GUIDE v2.5 02-Mar-2005 11:19:14
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TRIALMUN_OpeningFcn, ...
                   'gui_OutputFcn',  @TRIALMUN_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before TRIALMUN is made visible.
function TRIALMUN_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TRIALMUN (see VARARGIN)

% Choose default command line output for TRIALMUN
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes TRIALMUN wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = TRIALMUN_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

axes(handles.PlotGraph)
fplot('5*cos(x)',[-5 5])

%set(handles.PlotGraph,plot(x,y,'ks',3.14156)
set(handles.PlotGraph,'XMinorTick','on')
%xlabel(handles.PlotGraph,'y = 5 cos (x)')
grid on

% --- Executes on selection change in SelectionOpts.
function SelectionOpts_Callback(hObject, eventdata, handles)
%options = gaoptimset;


% --- Executes on selection change in CrossoverOpts.
function CrossoverOpts_Callback(hObject, eventdata, handles)
%options = gaoptimset;

% --- Executes during object creation, after setting all properties.
function SelectionOpts_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to SelectionOpts (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes during object creation, after setting all properties.
function CrossoverOpts_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CrossoverOpts (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function PopEdit_Callback(hObject, eventdata, handles)
function EliteEdit_Callback(hObject, eventdata, handles)
function GenEdit_Callback(hObject, eventdata, handles)
function EditStall_Callback(hObject, eventdata, handles)
function CrossEdit_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function PopEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PopEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function EliteEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EliteEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function GenEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to GenEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function EditStall_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EditStall (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function CrossEdit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CrossEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)

options = gaoptimset;

population  = str2num(get(handles.PopEdit,'String'));
elite       = str2num(get(handles.EliteEdit,'String'));
generation  = str2num(get(handles.GenEdit,'String'));
stall       = str2num(get(handles.StallEdit,'String'));
cross       = str2num(get(handles.CrossEdit,'String'));

%%%%SELECTION
sel_opts    = set(handles.SelectionOpts,'String');
if (strcmp(sel_opts,'Stochastic'))
    sel_fcn=@selectionstochunif;
else
    sel_fcn=@selectionroulette;
end

%%%%CROSSOVER
cross_opts  = set(handles.CrossoverOpts,'String');
if (strcmp(cross_opts,'SinglePoint'))
    cross_fcn=@crossoversinglepoint;
else if (strcmp(cross_opts,'TwoPoints'))
    cross_fcn=@crossovertwopoint;
else
    cross_fcn=@crossoverscattered;
end
```

```
%BEGIN FILL IN THE BLANK
if ( population<10 | population>100 )
   warndlg('Invalid POPULATION. The range ( 10-100 ) ','!! Warning !!')
else
   pop = population;
end

if ( elite<1 | elite>10 )
   warndlg('Invalid ELITE COUNT. The range ( 1-10 ) ','!! Warning !!')
else
   el = elite;
end

if ( generation<20 | generation>200 )
   warndlg('Invalid GENERATION. The range ( 20-200 ) ','!! Warning !!')
else
   gen = generation;
end

if ( stall<10 | stall>100 )
   warndlg('Invalid STALL GENERATION. The range ( 10-100 ) ','!! Warning !!')
else
   st = stall;
end

if ( cross<0.1 | cross>1.0 )
   warndlg('Invalid CROSSOVER FRACTION. The range ( 0.1-1.0 ) ','!! Warning !!')
else
   cr = cross;
end

%GenEdit_Callback(hObject, eventdata, handles);
%SelectionOpts_Callback(hObject, eventdata, handles);
```

```
%CrossoverOpts_Callback(hObject, eventdata, handles);

options = gaoptimset(options,'PopulationSize',pop,...
                     'PopInitRange',[-1;1],...
                     'EliteCount',el,...
                     'Generations',gen,...
                     'StallGenLimit',st,...
                     'CrossoverFraction',cr,...
                     'SelectionFcn',sel_fcn,...
                     'CrossoverFcn',cross_fcn,...
                     'PlotFcns' ,{ @gaplotbestf,@gaplotrange });

%%Fitness function
fitnessFunction = @simpleCos;
%%Number of Variables
nvars = 1;

%%Run GA Solver
[X,FVAL,REASON,OUTPUT,POPULATION,SCORES] =
ga(fitnessFunction,nvars,options);

set(handles.BestX,'String',X);
set(handles.BestF,'String',FVAL);
set(handles.Stopping,'String',REASON);

%axes(handles.time_axes)
%plot('5*cos(x)',[1 4])
%set(handles.PlotGraph,'XMinorTick','on')
%grid on

%fplot('5*cos(x)',[2 5]);

end
```

```
% --- Executes on button press in Info.
function Info_Callback(hObject, eventdata, handles)
% hObject    handle to Info (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Guide;

% --- Executes on button press in Help.
function Help_Callback(hObject, eventdata, handles)
% hObject    handle to Help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Help;

% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
initialize_gui (gcbf,handles);

function initialize_gui(fig_handle, handles)
data.population = 20;
data.elite = 2;
data.generation = 100;
data.cross = 0.8;
data.stall = 50;
setappdata(fig_handle, 'metricdata', data);

set(handles.PopEdit, 'String',20 );
set(handles.EliteEdit, 'String',2);
set(handles.GenEdit, 'String',100 );
set(handles.CrossEdit, 'String',0.8);
set(handles.StallEdit, 'String',50);
%set(handles.mass, 'String', 0);
```

```
set(handles.BestX,'String',' ');
set(handles.BestF,'String',' ');
set(handles.Stopping,'String',' ');

% --- Executes on button press in Exit.
function Exit_Callback(hObject, eventdata, handles)
% hObject    handle to Exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

selection = questdlg('Are you sure you want to exit?',...
                ['Exit ' get(handles.figure1,'Name') '...'],...
                'Yes','No','Yes');
if strcmp(selection,'No')
   return;
end

delete(handles.figure1)

function BestX_Callback(hObject, eventdata, handles)
% hObject    handle to BestX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of BestX as text
%        str2double(get(hObject,'String')) returns contents of BestX as a double


% --- Executes during object creation, after setting all properties.
function BestX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BestX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function BestF_Callback(hObject, eventdata, handles)
% hObject    handle to BestF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of BestF as text
%        str2double(get(hObject,'String')) returns contents of BestF as a double


% --- Executes during object creation, after setting all properties.
function BestF_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BestF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Stopping_Callback(hObject, eventdata, handles)
% hObject    handle to Stopping (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Stopping as text
%        str2double(get(hObject,'String')) returns contents of Stopping as a double


% --- Executes during object creation, after setting all properties.
function Stopping_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Stopping (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes during object creation, after setting all properties.
function PlotGraph_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PlotGraph (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate PlotGraph
%fplot('5*cos(x)',[2 5]);
%subplot(2,2,1);
```